

# Python量化交易

张杨飞 / 编著

電子工業出版社

Publishing House of Electronics Industry

北京•BEIJING



電子工業出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

## 内 容 简 介

本书本着能让新手快速上手量化交易的原则，循序渐进地讲解了量化交易入门所需要的知识，以及大量的开发技巧与交易技巧，具有很强的实用性。`vn.py` 是机构级别的量化交易软件，掌握 `vn.py` 框架原理并且熟练运用，有利于新手快速搭建属于自己的量化交易系统。`Python` 语言有非常强大的数据分析库，对于交易策略的研发具有天然优势，且其易学性也深受初学者喜爱。本书即以 `Python+vn.py` 这一流行组合写作，从量化交易的起源及其发展进程入手，在简单介绍 `Python` 量化编程基础，以及详细解析 `vn.py` 架构之后，全面地介绍了 CTA 策略、海龟策略，以及新策略的开发流程。

相对其他量化交易方面的书，本书不再讲述 `Python` 语言编程的大量细节，而将笔墨着重放在对量化交易策略的解析、应用与回测之上，这才是新手真正需要学习和实践的地方。本书适合所有对量化交易感兴趣的人员阅读，也适合相关院校和培训机构作为量化交易系统课程的教材。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有，侵权必究。

## 图书在版编目（CIP）数据

`Python` 量化交易 / 张杨飞编著. —北京：电子工业出版社，2019.5  
ISBN 978-7-121-36140-1

I. ①P… II. ①张… III. ①股票交易—应用软件 IV. ①F830.91

中国版本图书馆 CIP 数据核字(2019)第 046455 号

责任编辑：孙学瑛

印 刷：

装 订：

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编：100036

开 本：787×980 1/16 印张：26 字数：414 千字

版 次：2019 年 5 月第 1 版

印 次：2019 年 5 月第 1 次印刷

定 价：99.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 [zltts@phei.com.cn](mailto:zltts@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

本书咨询联系方式：010-51260888-819，[faq@phei.com.cn](mailto:faq@phei.com.cn)。



电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

# 前言

在证券交易领域中，量化交易脱胎于传统的主观交易：把投资者的交易理念、交易策略固化成计算机程序，通过算法快速自动下单，有效地防止投资者自身情绪的干扰，让其把精力放在研发交易策略上。量化交易的另一个优点是其成功的方法有迹可循，因为量化交易的过程是，运用现代统计学理论对历史数据进行数据分析，构建数学模型来预测市场未来价格的变化，然后通过计算机语言表达出来，从而实现自动交易。

vn.py 是机构级别的量化交易软件，掌握 vn.py 框架原理并且熟练使用，有利于新手快速入门量化交易，搭建属于自己的量化交易系统，也可以在机构中找到与量化岗位相关的工作。

## 为什么写作本书

本书使用的编程语言是 Python。尽管市面上关于 Python 量化交易方面的书籍不少，但是大部分着重讲述 Python 编程基础，而且主要是在股票交易中的应用，在期货市场的应用少之又少。就实践的层面来看，股票量化交易对于入门者几乎是不可能实现的，尽管 2019 年重新开放程序化交易 API，但是其资金门槛高达 5 亿元。

期货市场上程序化交易接口却无资金门槛，而且量化交易应用发展得比较成熟，这才是新手可以去学习和实践的地方。故本书的写作定位是交易策略在期货领域的应用和开发，力求填补这方面的空白。

## 本书特色

本书尽量以初学者的角度来讲述量化交易的内容，逐步填平量化交易入门要踩的“坑”，力求让读者快速熟悉这方面的知识，能够独立开发交易策略并且尝试进行仿真



电子工业出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

交易。如果在 SinNow 仿真交易平台能够赢利，那么就可以上实盘去“跑”了。

本书的另一个特色是使用机构级别的开源交易软件：`vn.py`。机构级别软件对应的使用群体是做量化交易的机构投资者，如私募基金、证券自营，以及资管和期货资管等。这类软件虽然上手困难，但是熟练掌握后能更有效地深耕于量化交易领域，并且也有利于初学者入门量化交易。

## 本书主要内容

本书共包括 7 章，每章的主要内容如下。

第 1 章“量化交易速览”首先从狭义和广义两个方面介绍了“量化交易”的概念，然后介绍开拓出这个领域的先驱们的故事，接着讲述量化投资在美国与中国的历史发展进程，最后简单地介绍国内常用的量化交易策略及宽客这个专门从事量化交易的职业。

第 2 章“Python 量化编程基础”介绍了将 Python 作为量化交易入门语言的理由，讲解了 Python 的基础概念，以及常用的数据分析库 NumPy 与 Pandas、机器学习库 scikit-learn，最后讲解绘图库 Matplotlib 的基本用法。

第 3 章“`vn.py` 入门”介绍了 `vn.py` 交易系统的概况、安装步骤、主交易界面的功能，具体讲述 `vn.py` 应用框架的结构，分别是底层接口、中层引擎及上层应用，最后对这 3 层结构的原理做一个具体说明。

第 4 章“在 `vn.py` 中实现 CTA 策略”介绍 `vn.py` 提供的数据库解决方案，用于生成具体 CTA 策略的相关支出模块，如 K 线生成、K 线管理和策略模板，最后讲述回测和优化模块。

第 5 章“经典 CTA 策略”主要介绍 `vn.py` 官方提供的经典 CTA 策略，包括策略原理、代码解释、策略回测和参数优化。

第 6 章“海龟策略本地化实证”首先介绍海龟交易策略的起源、关键要素，然后解析 `vn.py` 下海龟策略的代码，通过交叉检验与筛选品种构成投资组合，最后基于构



建好的海龟组合对策略的各个关键要素进行研究。

第7章“新策略实战”首先介绍了开发新策略的流程，然后是搭建投资组合，并进行策略回测为实战做好准备，最后介绍在真实交易情况下接触的3套系统，并且分析策略回测与实战中结果不同的成因。

## 致谢

我首先要感谢“猴子聊人物”创始人，他的数据分析的课程让我快速上手 Python 语言；然后是“用 Python 的交易员”陈晓优先生，我也是受益于其知乎 Live 上对量化交易的推广才从传统的金融转到该领域的。

我还要真诚地感谢电子工业出版社优秀的 IT 编辑孙学瑛女士和电子工业出版社对本书的重视，以及他们为本书出版所做的一切。

---

## 读者服务

---

轻松注册成为博文视点社区用户（[www.broadview.com.cn](http://www.broadview.com.cn)），扫码直达本书页面。

- **提交勘误：**您对书中内容的修改意见可在 [提交勘误](#) 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动：**在页面下方 [读者评论](#) 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：<http://www.broadview.com.cn/36140>



# 目 录

第 1 章 量化交易速览.....	1
1.1 为何选择量化交易.....	1
1.1.1 量化交易的概念.....	1
1.1.2 主观交易与量化交易.....	2
1.2 量化交易的先驱们.....	5
1.2.1 朱尔斯·雷格纳特.....	5
1.2.2 爱德华·索普.....	6
1.2.3 托马斯·彼得菲.....	9
1.2.4 詹姆斯·西蒙斯.....	14
1.3 美国量化投资的发展历史.....	17
1.3.1 兴起阶段（1970—1990 年）.....	17
1.3.2 快速发展阶段（1990—2000 年）.....	18
1.3.3 稳步增长阶段（2000 年至今）.....	19
1.4 中国量化投资的发展历史.....	20
1.4.1 ETF 套利时代（2010 年以前）.....	20
1.4.2 多因子 Alpha 和高频交易称雄时代（2010—2015 年）.....	21
1.4.3 多元化投资时代（2016 年至今）.....	23
1.5 国内常用的量化交易策略.....	24
1.5.1 期货 CTA 策略.....	24
1.5.2 股票 Alpha 策略.....	32
1.5.3 期权波动率套利策略.....	41
1.5.4 高频交易策略.....	45
1.6 宽客.....	48
1.7 宽客的两大阵形：P 宗与 Q 宗.....	51



1.8	宽客的 3 种职能分类 .....	52
1.8.1	量化 IT 工程师 .....	52
1.8.2	量化研究员 .....	53
1.8.3	量化交易员 .....	54
1.9	宽客的四大派系 .....	55
1.9.1	券商资管 .....	55
1.9.2	公募基金 .....	56
1.9.3	私募基金 .....	57
1.9.4	期货市场 .....	57
第 2 章	Python 量化编程基础 .....	59
2.1	Python 运行环境搭建 .....	60
2.1.1	安装 Anaconda2-5.0.0 (32 位) .....	61
2.1.2	设置 Anancoda 环境 .....	62
2.1.3	创建共享环境 .....	64
2.1.4	列出共享环境 .....	64
2.1.5	安装 Jupyter Notebook .....	65
2.2	数据 .....	66
2.2.1	字符串 .....	66
2.2.2	数字 .....	68
2.2.3	容器 .....	68
2.2.4	布尔值 .....	73
2.2.5	空值 .....	74
2.3	函数 .....	74
2.3.1	自定义函数 .....	74
2.3.2	第三方库的函数 .....	75
2.4	条件判断 .....	75
2.5	循环 .....	77
2.6	类和实例 .....	79
2.6.1	定义学生父类 .....	79
2.6.2	定义父类实例 .....	81



2.6.3	定义团体子类.....	82
2.6.4	定义子类实例.....	83
2.7	NumPy 与 Pandas .....	84
2.7.1	一维数组 .....	85
2.7.2	二维数组 .....	88
2.8	scikit-learn 机器学习库 .....	93
2.8.1	机器学习的步骤 .....	93
2.8.2	线性回归 .....	94
2.8.3	逻辑回归 .....	101
2.9	Matplotlib 绘图库 .....	104
2.9.1	用列表绘制线条 .....	105
2.9.2	用数组绘图 .....	106
2.9.3	多个图的绘制 .....	109
第 3 章	vn.py 入门 .....	111
3.1	vn.py 介绍 .....	111
3.2	搭建 vn.py 运行环境 .....	115
3.2.1	安装 Visual Studio 2013 社区版（特定版本） .....	115
3.2.2	安装代码编辑器工具：Sublime Text .....	116
3.2.3	安装 Wing IDE .....	117
3.2.4	安装 MongoDB 数据库 .....	117
3.2.5	安装 Robo 3T .....	120
3.2.6	安装 vn.py .....	121
3.2.7	更新 vn.py .....	123
3.3	VnTrader 界面功能介绍 .....	124
3.3.1	连接 CTP .....	124
3.3.2	界面说明 .....	125
3.4	vn.py 架构 .....	126
3.4.1	底层接口 .....	127
3.4.2	中层引擎 .....	128
3.4.3	上层应用 .....	129

3.5	底层接口 .....	130
3.5.1	CTP API 的工作原理 .....	130
3.5.2	CTP API 的 Python 封装设计 .....	135
3.5.3	CTP API 对接中层引擎原理 .....	137
3.6	事件引擎 .....	140
3.6.1	时间驱动 .....	140
3.6.2	事件驱动 .....	141
3.6.3	事件引擎工作流程 .....	142
3.6.4	事件引擎结构 .....	143
3.7	上层应用 .....	145
3.7.1	PyQt 介绍 .....	145
3.7.2	GUI 组件构成 .....	146
<b>第 4 章</b>	<b>在 vn.py 中实现 CTA 策略 .....</b>	<b>149</b>
4.1	数据解决方案 .....	149
4.1.1	CSV 加载模块 .....	149
4.1.2	开发新的 CSV 导入模块 .....	154
4.1.3	数据下载模块 .....	157
4.2	K 线生成模块 .....	159
4.2.1	1 分钟 K 线合成 .....	160
4.2.2	X 分钟 K 线合成 .....	163
4.3	K 线管理模块 .....	164
4.3.1	初始化参数 .....	164
4.3.2	生成时间序列 .....	165
4.3.3	定义属性函数 .....	166
4.3.4	生成计算指标 .....	167
4.4	CTA 策略模块 .....	169
4.4.1	定义成员变量 .....	170
4.4.2	构造函数 .....	171
4.4.3	回调函数 .....	172
4.4.4	主动函数 .....	173



4.5	策略回测模块.....	176
4.5.1	CTA 回测引擎.....	176
4.5.2	参数优化设置.....	180
4.5.3	调用回测和优化模块.....	180
<b>第 5 章</b>	<b>经典 CTA 策略.....</b>	<b>187</b>
5.1	双均线策略.....	187
5.1.1	策略原理.....	187
5.1.2	向量回测.....	188
5.1.3	vn.py 回测.....	193
5.2	Dual Thrust 策略.....	202
5.2.1	策略原理.....	202
5.2.2	策略代码解析.....	203
5.2.3	策略回测.....	208
5.2.4	策略优化.....	210
5.2.5	滚动回测.....	213
5.3	AtrRsi 策略.....	214
5.3.1	ATR 指标.....	215
5.3.2	RSI 指标.....	217
5.3.3	策略原理.....	218
5.3.4	策略代码解析.....	219
5.3.5	策略回测.....	222
5.3.6	滚动回测.....	223
5.4	金肯特纳通道策略.....	225
5.4.1	策略原理.....	225
5.4.2	策略代码解析.....	226
5.4.3	策略回测.....	231
5.4.4	滚动回测.....	231
5.5	布林带通道策略.....	233
5.5.1	策略原理.....	233
5.5.2	CCI 指标.....	234

5.5.3	ATR 指标 .....	236
5.5.4	策略回测 .....	237
5.5.5	滚动回测 .....	238
5.6	跨时间周期策略 .....	240
5.6.1	策略原理 .....	241
5.6.2	策略代码解析 .....	241
5.6.3	策略回测 .....	245
5.6.4	滚动回测 .....	246
5.7	多信号组合策略 .....	247
5.7.1	策略原理 .....	248
5.7.2	信号生成部分 .....	248
5.7.3	交易管理部分 .....	253
5.7.4	多信号策略的重构 .....	258
第 6 章	海龟策略本地化实证 .....	261
6.1	海龟策略速览 .....	261
6.1.1	海龟策略的故事 .....	261
6.1.2	海龟策略的局限性 .....	262
6.1.3	原版海龟策略 .....	263
6.1.4	策略回测效果 .....	268
6.2	本地化实现困境与解决方案 .....	270
6.2.1	本地化实现困境 .....	270
6.2.2	理想解决方案 .....	272
6.3	vn.py 实现的海龟策略 .....	273
6.3.1	工具准备 .....	273
6.3.2	数据准备 .....	274
6.3.3	海龟策略代码结构 .....	277
6.3.4	海龟策略 6 大要素代码解析 .....	280
6.3.5	海龟策略的回测 .....	285
6.4	品种选择验证 .....	287
6.4.1	原版投资组合测试 .....	287



6.4.2	筛选品种的传统方法 .....	288
6.4.3	构建海龟组合的难点 .....	297
6.4.4	海龟组合筛选的解决方案 .....	298
6.4.5	重新构建投资组合 .....	302
6.5	长短周期信号检验 .....	322
6.6	上一笔赢利过滤检验 .....	324
6.7	手续费、滑点测试 .....	326
6.8	单位头寸限制检验 .....	326
6.9	关于海龟策略的其他研究方向 .....	330
<b>第 7 章</b>	<b>新策略实战 .....</b>	<b>331</b>
7.1	开发新的策略 .....	331
7.1.1	策略思路 .....	331
7.1.2	增加 AROON 函数 .....	333
7.1.3	策略代码解析 .....	334
7.1.4	策略回测 .....	336
7.2	多策略的组合回测 .....	338
7.2.1	历史表现 .....	339
7.2.2	预测表现 .....	342
7.2.3	回测的注意事项 .....	342
7.3	模拟测试 .....	349
7.3.1	策略文件目录 .....	349
7.3.2	实盘/模拟盘配置文件 .....	350
7.4	真实交易环境 .....	353
7.4.1	交易环境的 3 套系统 .....	353
7.4.2	交易环境的数据流 .....	354
7.5	实际操作注意事项 .....	355
7.5.1	计算错误 .....	355
7.5.2	数据使用误差 .....	356
7.5.3	过拟合 .....	357
7.5.4	幸存者偏差 .....	358



7.5.5 策略周期 .....	359
7.5.6 动态变化的现实环境 .....	360
7.5.7 人为干预 .....	361
<b>附录 A 主流交易品种 .....</b>	<b>362</b>
A.1 股票 .....	362
A.1.1 股票的定义 .....	362
A.1.2 股票交易所 .....	363
A.1.3 股票竞价规则 .....	364
A.1.4 T+1 制度 .....	368
A.1.5 股票交易策略 .....	370
A.2 期货 .....	372
A.2.1 期货的定义 .....	372
A.2.2 期货交易所 .....	372
A.2.3 期货交易策略 .....	375
A.3 期权 .....	377
A.3.1 期权的定义 .....	377
A.3.2 期权的分类 .....	380
A.3.3 期权的影响因素 .....	382
A.3.4 期权投资组合 .....	384
A.3.5 期权波动率套利策略 .....	387
A.4 外汇 .....	388
A.4.1 外汇的定义 .....	388
A.4.2 外汇市场的结构 .....	390
A.4.3 外汇市场的组织形式 .....	393
A.4.4 主要外汇交易中心 .....	394
A.4.5 外汇交易策略 .....	396
<b>参考文献 .....</b>	<b>399</b>



# 第 1 章

## 量化交易速览

本章将带领读者迅速了解量化交易。首先，从“量化交易”概念入手，通过行业先驱故事来阐述量化交易的重要性，然后分别讲述量化交易在美国与中国的历史发展进程，最后简单地介绍国内常用的量化交易策略，以及“宽客”这个职业。

### 1.1 为何选择量化交易

#### 1.1.1 量化交易的概念

在回答“为何选择量化交易”这个问题前，先理解量化交易的概念。

严格地说，量化交易是运用复杂的统计学方法和数学模型，从庞大的历史数据中海选出能带来超额收益的多种“大概率”事件以找出规律、制定策略，并且能用数据模型验证、固化这些规律和策略，然后用计算机来严格、高效地执行之。

这一定义涉及统计学、金融学和计算机科学等多门学科，看起来高不可攀，但是，通俗点说，量化交易是指利用统计学、数学、计算机技术和现代的金融理论来辅助投资者更好地赢利。这些量化的方法可用于分析海量历史数据，也可用



于具体信号生成，或者控制持仓大小、进行风险控制等。

因此，我们可以惊喜地发现量化交易与传统的主观交易不再是二元对立，量化交易也包含主观交易，如比较常见的期货跨市套利策略、期权波动率套利等就属于半自动交易。它们需要交易员综合历史均值回归，以及对宏观政策的主观解读，开盘前调整好参数让计算机严格执行策略。

综上所述，量化交易是对主观交易的升华，剔除部分人为不稳定的因素，让用户专注于寻求超额收益（即 Alpha）。所以量化交易必然会成为历史发展的趋势。

## 1.1.2 主观交易与量化交易

量化交易总是被人们拿来与主观交易讨论，孰高孰低、众说纷纭。为了更好地分析它们之间的辩证关系，下面分别解读之。

### 1. 主观交易

运用主观交易的投资者需要关注国际环境及财经新闻、券商的研究报告、公司的财务报告、K线趋势、个股新闻、价量经验、大人物的演讲、朋友圈情绪、甚至一些小道消息，并对其进行定性或者定量分析。定性分析可以是体验公司产品或者直接实地考察，定量分析则可根据自己的选股原则，用各种指标进行打分，加权汇总后，买入分数高的股票，卖出分数低的股票，以形成交易信号。在手动下单方面，注意不要记错代码并且要避免“胖手指失误”<sup>1</sup>（fat-finger error），若单子太大还需要拆分成小单来降低成本。开仓之后要有风控意识并且严格遵循自己的止赢止损原则。

主观交易具有较强的主观能动性，即在同样的选股原则和止赢止损策略下，100 个人操盘会有 100 种不同的结果，基于大数定律和类正态分布，可以发现有一小撮人的资金曲线非常平滑和漂亮，但是绝大部分人都会低于均线水平，这里的“均线”指的是计算机严格执行交易策略所得到的资金曲线。原因往往是，理

---

<sup>1</sup> 即交易员在交易时因键盘操作失误而造成的输入错误。

性总是被情绪打败的。

20 世纪 60 年代，保罗·D·麦克莱恩（Paul D. MacLean）从生理角度提出了“三位一体的大脑”理论。此理论根据在进化史上出现的先后顺序，将人类大脑分成“爬行动物脑”“古哺乳动物脑”和“新哺乳动物脑”三大部分。每部分“脑”通过神经纤维与其他两者相连，但各自作为相对独立的系统分别运行，各司其职。“新哺乳动物脑”又称为边缘系统，它参与调解本能和情感行为，其主要作用是维持自身生存和物种延续。其中，杏仁核负责创造情绪并产生与之相关的记忆，海马结构能将短期记忆转化为长期记忆。杏仁核作为大脑的“恐惧中心”，它触发的保命技能可是顶尖的：远远看到狮子，恐惧会瞬间攫取你对身体的自主权并快速逃跑。杏仁核在漫长的人类演进过程中很好地提高了人类的生存概率，但是现在却阻碍了止赢止损策略的执行，让赢利变得困难。交易中，K 线的每一次跳动都会刺激着杏仁核，K 线越接近止损线，我们的恐惧越深，同时消极情绪的慢慢积累也会改变交易心态，为了解脱就选择提前离场，自我安慰着少亏就是赢，至少落袋为安。但是最后没想到行情随后一路上涨，一直接近原先设好的止赢线。蓦然回首，发现交易策略其实是对的，因情绪影响而没执行好策略。

在盯盘过程中，能够慢慢地读懂市场、看懂盘面，基于自己交易原则的基础上随机应变的人少之又少。天赋与运气缺一不可。这也就为什么在 A 股市场、期货市场 and 外汇市场这些偏短线投资的领域中，手动交易员如同大浪淘沙一样，换了一批又一批，能够稳定赢利的少之又少。“股神”巴菲特最厉害之处，未必是他的选股眼光，而是他控制“新哺乳动物脑”的功力。

## 2. 量化交易

量化交易不仅关注历史行情数据、基本面指标数据，而且会把一些非传统的数据，如市场情绪、财经新闻的关键字转化成机器可以理解的指标。数据越原始越好，例如可以直接购买交易所最原始的、未经清洗的行情数据。第三方数据提供商尽管物美价廉，但是在数据清洗过程中可能会把看似无用、实则隐含赢利机会的数据去掉。



有了数据之后，就需要运用数学和统计学的方法，如单位根检验、线性回归、机器学习等方法从大数据当中找到超额收益的多种“大概率”事件，比如选股的量化思想就是进行收益拆解，从很多维度进行数量化的判断。传统上把所有因子分成 7 大类：赢利性、估值、现金流、成长性、资产配置、价格动量和技术面，通过现代统计学的方法进行冗余因子的剔除和降维、因子权重的确定，以及对精英因子进行打分。基于选股模型的判断形成交易信号，通过程序化交易的 API 进行自动化交易，并且在交易系统上引入风险控制模块来管理持仓头寸。

量化交易的一大优点就是计算机的高效执行将人从简单重复的任务中解脱出来，可以把更多精力放在更好策略的开发上。另外，量化交易可以从更快速和更微观的维度去思考问题，人受限于人体的生理机构，从看到数据、大脑判断到手指敲在键盘的反应时间是几百毫秒，而计算机执行是可以达到纳秒级别的（1 秒 = 1 000 毫秒 = 1 000 000 微秒 = 1 000 000 000 纳秒）。因此在超高速领域，量化交易可以赚取主观交易无法赚到的钱。

任何事物都有两面性，正因为机器能够完美执行策略，公司需要严格保密各自的核心赢利策略。举个例子说明，若一个很赚钱的策略被泄露，致使很多公司都“跑”这个策略，结果就是只有少数速度快的公司能够赚钱，绝大多数是不赚钱甚至亏钱的。但如果大家都去提高速度，就会衍化成设备竞赛。对于每年上千万元级别硬件的维护和升级，不是每家公司都承担得起的。

主观交易者对策略思路的保密性要求并不高，就算是很好的策略让 100 个人执行，可能就两三个人赚钱，其他是亏钱的，这方面更看重人为的因素，如对宏观政策独特的见解、多年交易经验积累下来的盘感等。在对策略应用的随机应变上，主观交易也有出彩的地方，例如在期货市场上面对不断变动的报价版，资深主观交易者或许能够得知对手是“唯快不破”的高频交易公司，在琢磨透高频公司的交易思路后，通过超大订单去打击并猛追猛打，对手估计就不得不砍仓。

总之，不管是主观交易还是量化交易，到最后都能获得交易圣杯。不同在于主观交易的道路，看似平坦，其实走到越后面越陡峭无比，进化过程是玄之又玄

的，需要天赋、运气和顿悟。而量化交易更像现代意义上的搏击运动，它有非常系统的训练方案，只要打好基本功，一步一个脚印，总是能预见进步的。

## 1.2 量化交易的先驱们

本节主要介绍开拓量化交易这个领域的先驱们的人生经历，他们分别是最早使用量化投资方法的朱尔斯·雷格纳特、专业赌徒出身并且最早运用可转债套利的爱德华·索普、盈透证券创始人托马斯·彼得菲、大奖章基金创始人詹姆斯·西蒙斯。

### 1.2.1 朱尔斯·雷格纳特

从现存资料看，最早采用科学方法来研究和发现股票价格涨跌规律的人，既不来自股票和股市交易起源地的荷兰，也不是将金融实践发扬光大的英国人，更不是一开始就和金融共生在一起的美国人，而是看起来有些不“靠谱”的法国人。

最早采用量化方法来分析数据变化并从中挖掘市场价格涨跌规律的，是 1834 年出生于法国贝当的朱尔斯·雷格纳特（Jules Regnault）。雷格纳特出身贫寒，成年后到巴黎证券交易所成为股票经纪人助理，在跑腿送信的日常工作中耳濡目染着股票涨跌，强烈激发起他探寻规律并赢取股市财富的梦想。雷格纳特每天下班后坚持用纯手工方法，耐心细致地统计和梳理巴黎证券交易所 1825 年至 1862 年的股票、国债价格的数据，他惊奇地发现如果股票或国债的持有周期翻倍，价格偏差将增长 1.41 倍；如持有周期翻 3 倍则价格偏差增长 1.73 倍；翻 4 倍则增长 2 倍……由于 1.41、1.73 和 2 分别是 2、3、4 的平方根，这令雷格纳特激动异常：“我首次发现了一个未被人表达的数学自然规律，股票价格的差值与所考察的股票持有周期平方根成正比。”

雷格纳特发现了债券的错误定价，运用类似“统计套利”的方法来交易 3% 永续国债（面值 100 法郎，每半年付息 1.5 法郎且永远不偿付本金），根据该国



债价格长期浮动于 32.50 法郎至 86.65 法郎之间，计算出其合理价格应为 73.4 法郎。低于此价他就果断买进，偏离越远买进越多；高于此价就果断卖出，偏离越远卖出越多。就这样，虽然也会遭遇短期的缩水和亏损，但持续到 1881 年，47 岁的他实现了“财务自由”。他购买了庄园，雇了车夫、园丁，买了 3 辆马车和几匹好马，每年都去度假。1894 年，雷格纳特去世，留下由债券、股票和房产构成的 300 万法郎的财富。

随后出版的《概率计算和股票交易哲学》标志着量化交易的开始。而真正“轰”开量化世界大门的是随后一批“不务正业”的“学霸”们。

## 1.2.2 爱德华·索普

爱德华·索普在小时候就对数字显示出了极大的热情。由于家境贫困，索普很早就开始想办法挣钱。有一次，索普和杂货店老板打赌看谁能以最快的速度算出账单金额，他用大脑，杂货店老板用计算器，结果爱德华·索普取胜，并获得了冰激凌蛋卷作为奖品。有时，索普会花五分钱买一盒饮料，再往里面掺些水，按一分钱一盒的价格卖给口渴的工人，这样每盒饮料可以赚一分钱。

1955 年春天，索普在加利福尼亚大学洛杉矶分校（UCLA）的物理系读研究生时，和同学们讨论什么方式能不劳而获，大家提到了赌博，而讨论的最终结果是，赌博永远赚不到钱。但索普认为如果能从数学角度找到赌博的密码，那么就一定能战胜庄家。于是他先用计算机预测轮盘赌会出现的每个数字，但是他发现机器的预测性太差，就放弃了。这时他看到了一篇关于计算二十一点扑克游戏的最佳策略的论文，就去赌场实践了，结果输得很惨。

直到他当上麻省理工学院的数学老师之后才研究出二十一点的制胜秘诀，并写了一篇文章准备介绍他的秘诀。索普打算把这篇论文发表在著名杂志《国家科学院文献》上。但根据规定，论文必须由国家科学院的院士递交才能发表。当时麻省理工学院的唯一一位研究数学的国家科学院院士就是克劳德·申农。于是，索普给申农的秘书打了电话，预约面见他。两位量化先驱就这样相遇了。申农建议





他把论文名字确定为《二十一点的制胜秘诀》。

论文发表后，信件如雪片般飞向索普，向他讨教二十一点的制胜秘诀，以至于麻省理工学院院方不得不禁止索普对此再发表任何言论。但是索普和申农决定把理论上升到实践，他们找到几个出资人资助他们去拉斯维加斯一试身手。

从拉斯维加斯回来之后，索普写了一本书，即《战胜赌场》，这本以数学研究为主的书，因为名字而成为畅销书。通过索普的书，一时间拉斯维加斯的赌场生意兴旺，但是学会索普的下注方法的人依然不多。拉斯维加斯的赌场已经提醒门卫注意索普这个长相的人出入。索普要像好莱坞电影《决胜21点》的男主角那样，每次进出赌场都不得不乔装打扮，转换说话口音来迷惑赌场门卫。1964年，在拉斯维加斯的一家赌场玩到晚上9点时，索普要了杯咖啡。结果喝下去之后，索普发现自己难以集中注意力，眼睛什么也看不清楚，只好摇摇晃晃地从牌桌边起身回到旅馆房间，过了8个小时才恢复正常视力。第二天，索普又来到这家赌场，这次只要了白水，而且每次只喝一小口。“那水味道很怪，好像里面放了一大包发面用的苏打粉，只用几滴就能让我昏睡一整夜。我要是再多喝一点肯定会立刻在牌桌前昏过去。”幕后黑手毫无疑问是恨他入骨的赌场老板。这件事让索普十分后怕，他决定将自己的目光转向另一个方向——股市，在他看来，这是另一个赌场。

索普研究的第一个标的是股票权证。这种权证和看涨期权一样，赋予购买者在未来以某个价格购入股票的权利，主要的交易渠道是做市商，是当时投资客们的最爱。

索普发现，运用大数定律，不能说每一只股票在明天是涨还是跌，但是可以推算股票涨跌一定幅度的概率。也就是说，如果股票的波动率是随机的，那么它就是可以量化的。这个发现，为量化交易奠定了基调。

而对于那些股票权证而言，通过对波动率的估计，我们就能知道其定价是否失准。比如你买了一份贵州茅台的权证，它的现价是100元，6个月后需要上涨到120元你才能赚钱，那么你就需要计算在此期间上涨到120元的概率是多少，





并基于这个概率来得到这份权证的价格。索普利用随机游走模型，再加上某只股票比其他股票是否上涨或下跌更大比例的一个变量，就可以估算这份权证到底值多少钱。

索普为此找到了一名叫席恩·卡索夫（Sheen Kassouf）的金融学家一起合作，把市面上所有的股票权证都测了一遍，结果让他们非常开心，因为在他们的模型下，这些权证的定价基本都是错误的。利用这些错误的定价，他们卖空高价权证，用股市的相应股票作为对冲，如果股票出人意料地上涨，那么股价的涨幅可以弥补他们在权证上的损失，而公式会告诉他们买入多少股票是正确的。索普的权证定价模型就是 Black-Scholes 期权定价模型的最早实践。

这其中还有一段趣闻：麻省理工学院的布莱克（Black）和斯科尔斯（Scholes）花了相当一段时间才整理好论文准备出版。当一切就绪时，布莱克发了一份预印的材料给爱德华·索普，认为他可能会对此感兴趣。他在材料后附的信上解释说他把索普的推理又向前推进了一步。在一个完美的理性世界，没有什么比无风险投资更有价值了。避险对冲理论上就是无风险投资，因此应该像国库券那样的无风险投资一样带来相同的利润，当然前提是期权的价格定得“准确”。索普迅速地将布莱克和斯科尔斯的公式编辑到计算机中，做出了一个定价表格。他把这个表格和自己做的公式进行了比对，发现除无风险利率的指数因子外，两者几乎完全相同。公式很快被命名为 Black-Scholes 期权定价模型，于 1973 年正式出版。当然，这个名字并没有显示出索普曾经为此所做过的努力。“事实上，我从来没有在乎知名度。”索普说，“因为我不从事经济学和金融领域的工作。我觉得知名度这个问题并不重要，我更关注的是如何多赚钱。”后来，布莱克、斯科尔斯在 1997 年获得了诺贝尔经济学奖。

索普是一个很大方的人，他曾告诉席恩·卡索夫（Sheen Kassouf）应该把这无风险的对冲体系展示给整个世界。于是《战胜市场：一个科学的股票市场系统》（*Beat the Market: A Scientific Stock Market System*）就诞生了，而这本书也成为量化投资领域的开山之作。书中向小规模投资者描述了一种很简单的权证对冲体系。当时一般的家庭还没有计算机，只能用图纸画出表格来确定标价过高的权证。



索普在考虑未来的走向时，希望对资金进行专业化的管理。索普想要开始一种“市场中立”的投资合作关系，这意味着投资收益独立于股票市场的波动情况。不管股市表现怎样，都能有保持很好的业绩回报，合作关系也不会受到影响。这种合作关系就是后来大家所熟悉的追求绝对收益的对冲基金。

于是，1969年，索普和莱根成立了普林斯顿一新港公司，意即东西海岸之间的合作。索普和一名员工在加利福尼亚州负责数学问题，他们为东海岸的莱根和其他员工提供交易指导。东海岸的分公司负责打理一些商务上的事情，包括员工的招聘等。索普和莱根将他们这种新型的对冲基金合作关系称作可兑换对冲基金协会。“可兑换”指的是可兑换债券，这是索普发现的一种新机会。可兑换债券是由债券和权证组成的，当公司股价上升时，可转债的持有人有权利把手中的债券换成股票。

1970年，标普500下跌了5%，而索普的基金赚了3%，1972年，他的回报率是26%，比标普500高了11.7%。索普的基金成立之后连续11年获得两位数的回报。索普当年玩二十一点时，金主们给他的10000美元，到1969年基金成立时已经变成140万美元，他把这些钱作为基金的启动资金。到1985年，基金的规模是1.3亿美元。似乎没有什么能阻挡索普赚钱，也没什么能阻挡他成为华尔街竞相模仿的偶像。

1987年爆发了“黑色星期一”这个“27个标准差的事件”，“黑色星期一”成为所有宽客心中的噩梦，是对所有人相信的概率原则的一次挑战。但是索普如同奇迹一般地在这次股灾里毫发无损，1987年他的基金赢利了27%，“黑色星期一”里，他通过后续的反向操作弥补了大跌带来的亏损，最后只亏了几百万美元，就他基金的规模而言，简直是奇迹。

### 1.2.3 托马斯·彼得菲

1944年，二战期间，托马斯·彼得菲出生在匈牙利首都布达佩斯一家医院的地下室里。他的童年充满了饥饿、困苦，经常为了躲债而不得不跟着父母半夜搬



家。1965 年，他抓住了逃离匈牙利的机会，几乎身无分文地来到美国纽约，一个基本不懂英语的匈牙利人开始了他在美国华尔街的传奇淘金之旅。

初到纽约时，彼得菲不会讲英语，他在同是匈牙利人房东的帮助下找到了一份在一家工程公司做制图员的工作。他的公司买了第一台计算机时，由于当时计算机刚刚普及，公司中没有人会使用它。对于彼得菲来说，学习电脑语言要比学习英语简单多了，于是他毛遂自荐地学习编程，不久他便编写出了一个简单的勾股定理的算法——利用正弦或余弦函数计算，帮助工程师得出道路的半径和斜坡。彼得菲帮公司设计出了一套数据库并得到了每周 65 美元的薪酬，成为当时为数不多的程序员。

1967 年，彼得菲跳槽到为华尔街客户建立计算机系统的 Aranyi 公司。在 Aranyi 公司期间，彼得菲又编写了一个可以帮助投资者快速比较不同股票特点和价值的算法交易代码，在 Aranyi 公司工作的三年时间，使得彼得菲对金融市场开始了解，并成为当时华尔街为数不多的程序员中的佼佼者。

1969 年，彼得菲跳槽到纽约金融市场知名人士亨利·杰里克创办的莫卡塔公司。在莫卡塔公司里，彼得菲如鱼得水，得到了杰里克的赏识，他的编程技术也得到了发挥，在同年创造出华尔街首个所谓的黑盒——一个读取行情数据，经过一系列的算法处理后，产生交易买卖指令的系统。正是通过这个系统，莫卡塔公司取得了不错的收益。20 世纪 70 年代初，莫卡塔公司也开始参与期权交易，期权交易最难的是期权定价，当时还没有 Black-Scholes 公式可供参考。于是凭着交易员的感觉，杰里克和彼得菲总结出了当时影响期权定价的几个因素：期权行使价、期权到期日、波动率及无风险利率。彼得菲费尽周折、绞尽脑汁花了一年多的时间，编写出来一套运用所有上述参数来定价期权的算法。莫卡塔公司开始使用这个算法来交易期权，并取得了不错的业绩。后来他们发现这套方法与一年后发表的期权定价公式——Black-Scholes 模型类似。

到 20 世纪 70 年代末，随着芝加哥交易市场的建立，彼得菲认为这是股票期权市场兴起的标志，而且股票期权将是一个更大的交易市场，有更大的利润空间。

1977年，彼得菲积蓄了20万美元，花了3.6万美元在美国股票交易所买下了自己的交易席位，从此开始创业之路。初期彼得菲的期权交易进行得并不顺利。他的交易完全依靠自己算法提供的定价，如果一个期权合约没有在他保守的利润区间内，他是绝不会交易的。尽管他交易得如此小心，还是免不了失手，曾经一下子损失了十万美元。

尽管这次惨痛的教训来得如此突然、如此凶猛，但彼得菲没有被打倒，他重新振作，一步一步慢慢地积累资金，扩大交易团队，扩大交易量。但由于他的英语不好，他无法像其他交易员们一样和市场庄家们闲聊、交流，庄家们也不喜欢同这个说英语带有浓重口音的匈牙利人打交道，因此他的单子很多不能被庄家成交。于是，彼得菲有了个令人意想不到的想法来增加自己交易的成交比例。当时的美国金融市场是男人统治的天下，彼得菲却反其道而行，雇用了身材诱人的金发女郎来做他的交易员，反正这些金发女郎只是按照他的程序指令来交易，不需要任何交易盘感。这一招却让彼得菲一直不乐观的报单流发生了戏剧化的转变。市场庄家们的眼球一下子就被这些金发女郎们吸引。自然而然地，她们的单子也很快地被庄家们全部接收。这些庄家根本没意识到自己做的每一笔交易都在赔钱。

考虑到基于自己的算法程序，更高的成交量可以降低风险并增加收益，彼得菲决定在美国证券交易所加入做市商的行列。在彼得菲利用他的算法和金发美女创造利润的半年后，这些和彼得菲交易的庄家们和场内交易者开始注意到自己几乎就没有赢过彼得菲的美女杀手们。从技术上讲，无论市场往哪边波动，做市商需要持续保持买卖两边的报价。可是彼得菲扭曲了这条规则，他只挑出值得做的交易。庄家们早就受够了持续被彼得菲击败的滋味，警告他必须在一部分期权上同时保持买卖两边的报价，否则就取消他的做市商资格。可是要保持期权买卖两边的报价就要求他的交易员时刻关注市场的走势，这对一直以来依靠计算机算法的彼得菲团队来说是不现实的。他也不能要求他的交易员一直都紧贴着电话等待下一个交易指令。彼得菲的交易再次受阻，他又开始寻找新的解决方案。最终一个埋藏已久的想法呈现在他脑海：掌上电脑。经过一系列和美国证券交易所官方的争辩，交易所只允许彼得菲的交易员们带小型便携式电脑进场。可是最大的



问题是，当时根本没有什么便携式电脑。

带着一个朦胧的想法，彼得菲找到纽约大学的物理博士们来帮忙设计。他们在一个大小为 8×12 英寸、厚度 2 英寸的黑色盒子里，嵌入晶体管和集成电路板，用金线连接起电路板和顶端屏幕，做成了一部触摸式的平板电脑。这些平板电脑通过电话线连接 Quotron 行情数据机器接收行情数据，经过算法程序计算后，产生交易指令，并以短波无线方式回传到平板电脑上，那些美女交易员们拿到指令后再报价到庄家。然而，此时彼得菲的竞争对手使用的却是每天只能更新一两次的公平价值定价单。

从那开始，彼得菲赚到了超过 100 万美元一年的丰厚利润。为了扩大自己的业务规模，他考虑进军芝加哥期权交易所（美国最大的期权交易所）市场，但因交易场所禁止携带电脑之类的设备进入遭到了拒绝。最终彼得菲转到了纽约证券交易所扩大他的股票期权业务。为了更好地帮助他的交易员们提高效率，彼得菲设计出一些带光的彩棒，系统产生的不同交易指令，会通过不同的颜色示意给场内的交易员们。

慢慢地，彼得菲的交易量越来越大，赢利也开始直线上升，1986 年，交易所变成了彼得菲交易大军的取款机。公司从 1986 年 100 万美元的初始资金，到年底的 500 万美元，取得了 400% 的年回报。同时他在股票期权上的交易，必须在股票市场进行对冲，让他在股市的交易量也直线上升。

1987 年的一天，一位纳斯达克工作人员到彼得菲在世贸中心的办公地例行检查，查看他的交易运行，他预想的是骚动的人群、吵闹的电话声、打印机的声音，还有交易员向纳斯达克交易终端输入交易指令时此起彼伏的叫买叫卖声。可这场景并未出现在他眼前。实际上，他只看见一台 IBM 电脑连接着纳斯达克终端机，而且仅靠这台 IBM 电脑来完成彼得菲如此庞大的交易量，顿时感到不可思议。电脑里有指示交易品种、交易时间和交易数量的代码。这位纳斯达克职员没有想到，他刚刚见到的是世界上第一台运行全自动算法交易系统的电脑。彼得菲的设备可不只是像过去的交易系统那样提示交易品种，也不仅是简单地弹出需要人来执行

的交易单。这台电脑悄悄潜入纳斯达克交易终端，全权决定并执行交易，不需要人的参与。彼得菲的代码可以利用从纳斯达克终端不断涌来的交易数据，分析市场，轻易通过买家出价和卖家售价的不同来开出买单和卖单。这里值得一提的是，彼得菲的交易机构开启了华尔街的新篇章。从此，电脑程序员、工程师和数学家开始了对金融市场长达20年的大举进攻，所使用的利器就是算法和自动化交易，算法有时无比复杂精密，智能到几乎可以取代人成为金融市场的决定性力量。

当纳斯达克工作人员得知彼得菲通过外接或者更恰当地称此为“偷取”纳斯达克行情数据来实现IBM电脑的自动化交易时，他立即要求彼得菲停止这样的运作，因为纳斯达克的软件是给人工看行情且手工键盘输入交易指令用的，不是用来做自动化交易。

在被纳斯达克要求停止外接行情数据，且交易必须通过纳斯达克终端键盘执行后，彼得菲在之后的一周时间里和他最好的工程师忙着焊接金属、编写代码、焊接数据线，创造了一种输入交易和指令的机器手。当一星期后纳斯达克人员回来复查，他仿佛置身于科幻小说所描述的场景。交易单不断涌入，手柄噼里啪啦地打在键盘上，噪声如此之大，甚至淹没了谈话声。机器每次停下来，仿佛要安静一会儿，谁知转瞬之间再次启动，更加气势汹汹地弹出比上次还要多的单子。整个交易是华尔街的聪明人对规则的又一次令人叹为观止的绕行。最后纳斯达克人员摇摇头，彼得菲扮了个鬼脸。正是这套自动化交易系统帮助彼得菲在1987年一年里赚到了2500万美元。之后彼得菲设计了更多的系统和算法，进行了许多技术创新，进入了更多的交易品种和市场。他尤其喜欢那些新的市场，因为这些市场存在更多的机会，而且竞争也少。

1990年，彼得菲将他的公司改名为盈透证券（Interactive Brokers，简称IB）。在盈透证券中，工程师就是公司的一切，公司内75%的人员是程序员和工程师。

1993年，彼得菲将他日渐完善的交易系统作为服务，提供给客户，因为价格便宜，交易品种多，地域全，受到了极大的欢迎，尤其适合专业的交易公司和职业交易员。现在按公布的每日平均收入交易衡量，盈透证券已成为最大的网络经纪商。





2007 年 5 月 4 日，盈透证券在纳斯达克成功上市。在开市的钟声敲响后，他的公司被市场估值为 120 亿美元，成为当年美国第二大的 IPO。虽然盈透证券最低注资要求是 10 000 美元且不适合休闲投资者，但在国际交易和专业交易者所渴望的低成本佣金领域，盈透证券处于行业领先水平，而且盈透提供一个全能账户，客户可以在 24 个不同国家的 100 多个市场中心交易股票、期权、ETFs（交易所交易基金）、期货、外汇、债券，以及 CFD（差价合约）。

### 1.2.4 詹姆斯·西蒙斯

詹姆斯·西蒙斯（James Harris Simons）于 1938 年出生在美国波士顿郊区一个犹太家庭。据说，西蒙斯很小就开始表现出对数字、形状超出年龄的兴趣，他说自己 3 岁就想学数学了。西蒙斯在波士顿附近一个叫牛顿的小镇读完中学后，就进入了麻省理工学院读数学。他当年的导师数学家辛格回忆说：“西蒙斯的悟性很好，他能直观地感受到数学的原理，这是很罕见的。”

1958 年，20 岁的西蒙斯用了 3 年学完本科，转投加州大学伯克利分校攻读数学博士学位。期间他第一次结婚，婚礼收到的礼金都被西蒙斯拿去投资了，不管股票还是大豆期货都赚了。但是那时候西蒙斯还对投资、交易之类的事情兴趣不大。

三年以后西蒙斯就拿到了博士学位，回到母校麻省理工学院当老师，那时他才 23 岁。他的博士学位导师回忆说“吉姆（西蒙斯的昵称）是个很有独创性的人，喜欢坚持走他自己认定的路。”他的博士论文讨论的有关多维弯曲空间里的几何问题，和后来以西蒙斯和华裔著名数学家陈省身联合命名的 Chern-Simons 理论一样，都属于拓扑几何学的范畴。

1967 年，西蒙斯接受了成立才 10 年的纽约州立大学石溪分校（简称石溪大学）的校长陶尔的邀请，出任该校数学系主任。他招兵买马，不分远近，请到了一些当时或者后来的学科领军人物，这些人分别来自德国波恩大学、美国密歇根大学、俄罗斯的圣彼得堡国立大学，等等。西蒙斯在石溪大学待了 8 年，在这期

间，他不仅和加州大学伯克利分校微分几何学的顶尖人物陈省身共同创立了著名的 Chern-Simons 理论，而且使这个新成立大学的数学系的拓扑几何研究在全美名列前茅。

1974 年，西蒙斯在商品市场赚了钱后（从 60 万美元变成了 600 万美元），让他开始对交易有了兴趣。虽说他在学术上的成就已独领风骚，但他还是开始慢慢地移情别恋。后来他自己说“学术界的节奏太慢了”。而另外一个原因是西蒙斯研究的数学领域出成果很难，需要很长的时间，有时候还要靠相当的运气。

1978 年，他完全脱离了石溪大学，成了专业投资人。他成立了一个林姆若伊基金，专门从事各种投资活动，其中主要是外汇交易，但是也包括投资各种小公司现在被统称创投基金的投资活动。10 年间，林姆若伊基金的投资回报是 25 倍，相当于每年增长 38% 左右。

西蒙斯于 1988 年关闭了已有 10 年交易历史的林姆若伊基金，成立了文艺复兴科技公司，该公司管理的就是后来众所周知的大奖章基金。公司的地址不在金融界聚集的华尔街，而是在石溪大学的边上。此时西蒙斯的投资方法才从主观交易完全转型到量化交易。

大奖章基金和林姆若伊基金有两个明显的不同。第一个不同点，大奖章的投资范围不再包括创投基金。虽说西蒙斯的第一桶金源于投资小公司，而且他一生都对直接投资各种小公司有着浓厚的兴趣，但是大奖章基金投资的产品按照他本人的话来说必须符合三个条件：“必须在公众市场上交易；必须有足够的流动性；必须适合用数学模型来交易。”

第二个不同点，大奖章基金的投资方法是纯粹的量化投资，以技术数据为主，而林姆若伊基金的投资方法则以基本面数据和主观判断为主。对于其转型的原因，西蒙斯也回答道“首先，数学模型降低你的投资风险。其次，数学模型降低你每天所要承受的各种心理压力。”后面一点是很重要的，因为判断型的投资完全依赖大脑根据最新的信息做出最新的判断，所以，要想不贻误战机，大脑必须随时





随时随地保持高度警觉的状态，因为新的信息在不断出现，投资的仓位需要不断地调整。

开张第一年，大奖章基金赚了 8.8%。但是 1989 年起模型似乎开始罢工，从年初到 4 月，大奖章基金赔了 30%。西蒙斯花了 6 个月的时间冥思苦想，最后决定将过去模型中的有关宏观经济数据的部分完全剔除，只留下技术数据。同时，公司将注意力集中在短线交易上。这应该算是大奖章基金的重要转折点，当时制定的投资战略被保留至今，也是大奖章基金长盛不衰的立命之本。

新公司创立之初，西蒙斯的干将主要来自 3 个地方：一个是石澳大学的数学系，过去他曾经是系主任；另外一个国防分析研究院；第 3 个地方可能会令人感到惊奇，是 IBM 公司的语音识别实验室。有人曾经说，当年西蒙斯把整个语音识别实验室的精英统统都给挖走了。

有了这样一群形形色色局外人琢磨不透的科学家，公司的数学模型在不断更新和变化，很多的灵感就是在这样的跨学科碰撞中产生的，西蒙斯的大奖章基金也就理所当然地从胜利走向胜利。1994 年，美联储 6 次升息，利率从 3% 升到 5.5%，当年政府债券的收益为 6.7%，大奖章基金赚了 77%，2000 年的科技股股灾中，标准普尔 500 指数跌幅超过 10%，大奖章基金获得空前的丰收，净回报 98.5%。似乎每当股市或者债市越差，市场的波动性越大的时候，大奖章的表现就越好。西蒙斯自己也说过，他的基金需要一定的波动性才表现最好。他说：“要赚钱，就要市场动。”

大奖章基金的投资组合包括上千种不同的股票和各种其他金融工具，它的交易非常频繁和迅速，很多人在描述大奖章基金的交易方式时都会形容成“像机关枪一样”，基金过去每年的周转率在十几倍到几十倍之间。2003 年年底大奖章基金资金的股票投资金额为 82 亿美元，持有 1387 只股票，前一年年底的股票持仓量为 123 亿美元。它所持有的股票并不是那些很偏门的公司，而是交易量很大的生化、食品、医药、采矿、国防、金融行业的股票。

2005 年，文艺复兴科技公司的财务总监马克·西尔伯告诉外界，大奖章基金

已经达到了流动性的限额，所以将退还基金中外界投资人的钱。其实在之前的 3 年中，大奖章基金一直在不断返还外界投资人的钱。2005 年年底，外界投资人在大奖章里的钱已经被完全退还。2006 年开始，大奖章基金的所有投资人都是复兴技术公司的现任或者前任雇员及其家属，这样做的目的是使大奖章基金的总值保持在 50 亿美元左右，光是雇员的钱就已经达到这个数字。

但是文艺复兴科技公司并没有对外界投资者完全关闭大门，它新开设了两个基金来欢迎新的投资者。这两个基金的最低投资额为 2000 万美元，所以它主要的目标是机构投资者，尤其是退休金的管理公司。新的基金规模上限是 1000 亿美元，只投资美国的上市股票，所以和大奖章基金不同，后者买卖很多外汇和商品期货。新基金的投资时限也要比大奖章基金长，并且以买进股票为主，所以更接近公募基金。

## 1.3 美国量化投资的发展历史

美国量化投资的发展历史进程大致可以分为 3 个阶段，分别为兴起阶段、快速发展阶段，以及稳步增长阶段。

### 1.3.1 兴起阶段（1970—1990 年）

20 世纪 70~80 年代，量化交易逐渐在华尔街兴起，其重要标志是 1973 年，芝加哥期权交易所成立，Black-Scholes 期权定价公式开始被华尔街迅速接受。因为期权定价公式属于解析法，与二叉树模型（计算美式期权）或者蒙特卡洛模拟（计算特异期权）不同，它可以在交易系统底层非常快地计算出欧式期权理论价，当市价在一定程度上偏离理论价时，用于快速套利。因此期权高频套利一下子发展起来了。

当然，期权的贡献不仅于此：由于可以进行做多看跌期权这个操作，即当股价下跌时赚钱，而且跌得越多赚得越多，但是期权的价格只有股票价格的十几分



之一（期权内置杠杠倍数高）。在一篮子股票中买入看跌期权，相当于用极少的钱去买保险，牺牲现在一少部分钱来挽留住未来可能遭受的大幅度损失。有了期权，可以很方便地对冲整体市场风险，再通过基金经理的选股能力去获得超额收益，故期权也间接催生了市场中性策略。

1983 年，时任摩根士丹利大宗交易部门程序员的格里·班伯格（Gerry Bamberger）在为大宗商品交易部门编写软件的“配对策略”中无意发现“统计套利策略”。“配对策略”即买入一种股票的同时卖出另一种同类的股票以规避风险。班伯格的发现是大宗交易通常会导致这一对股票中的一只股票强烈波动，而另一只股票却几乎原地不动，即这一对股票的价差会暂时出现异常。班伯格基于这个发现提出了交易策略，在价差恢复到历史水平的时候获利平仓。该策略的强大之处在于不管市场运动何方都能够赢利，因此该策略大火，各大投行和对冲基金争相模仿。同样的策略，用的人多了赚钱难免就变难了，林子就这么大，一下子来这么多人采蘑菇，那肯定速度越快的人采到的蘑菇才越多。为了更快地抢到单子，各大机构投入重金放在优化硬件设备和算法上，开始了“军备”的竞赛，这从另一个层面也拉近了金融和自然科学界的距离。

### 1.3.2 快速发展阶段（1990—2000 年）

20 世纪 90 年代，美国经济总体而言是非常繁荣的。从 1992 年到 1999 年，美国经济平均每年增长 4%（从 2001 年起就再也没有超过 4%；从 2005 年起，全年增长率就再也没有超过 3%），每年平均增加 170 万个就业机会，约增长 85 万个就业机会。到 20 世纪 90 年代末，美国失业率从 1992 年的 8%降低到 4%，股价翻了四倍，道琼斯工业平均指数增长了 309%。

在这样的大时代背景下，量化投资也迎来了“黄金十年”。到 20 世纪末，互联网开始大量普及，依托于计算机编程的量化投资技术席卷市场。至此，对冲与量化终于结合起来，全球量化对冲基金市场迎来了春天。

这可谓是群雄崛起，很多金融界新秀在这一阶段一步登天、成就伟业！

1990 年，哈佛大学经济学本科生肯尼斯·格里芬（Kenneth C. Griffin）在索普的帮助下，设立大本营投资集团，利用数学模型进行可转债套利交易，后发展高频交易策略等，成为多元化多策略基金。第一年回报率达到 70%。2004 年基金资产达到 150 亿美元，2003 年格里芬个人资产高达 7.5 亿美元。

1991 年，普林斯顿大学数学系毕业生彼得·穆勒（Peter Muller）在由伯克利经济学教授巴尔·罗森堡（Barr Rosenberg）创立的 Barra 量化基金中发明了阿尔法系统策略，后到摩根士丹利成立过程驱动小组，由数学、统计与计算机界的顶尖高手组成，建立了“大富翁”程序化交易系统，专注于特定行业的统计套利。1996—2006 年利润约为 50 亿美元，团队奖金约为 10 亿美元。

1992 年，获得沃顿商学院本科金融与计算机双学士学位、芝加哥大学金融博士学位的克里夫·阿斯内斯（Clifford Asness）发明了价值和动量策略。他进入了高盛资产管理公司从事量化业务，建立了全球阿尔法基金，第一年收益率为 95%，第二年收益率为 35%。

1994 年，约翰·梅里威瑟（John Meriwether）从所罗门兄弟公司辞职，成立长期资本管理公司，斯科尔斯和莫顿加入。该基金的专长是相对价值交易，搜寻价格行为反常的证券，擅长用大杠杆买入老债券和卖出新债券。1998 年出现的“黑天鹅事件”，导致长期资本管理公司破产。

1995 年，博阿兹·温斯坦（Boaz Weinstein）从密歇根大学哲学系本科毕业进入美林公司全球债券交易部，1998 年进入当时世界最大银行德意志银行从事衍生品交易，专长于浮动利率债券和资本结构套利。2005 年内部基金规模实现 200 亿美元。

### 1.3.3 稳步增长阶段（2000 年至今）

自 2000 年以来，各种金融工具与计算机程序高度结合，带动量化对冲基金的快速发展，全球量化对冲基金市场高速增长。天天基金研究中心发表的《专题研究：量化对冲策略及产品简介》指出，2008 年金融危机前期，全球对冲基金规模



由 2000 年的 3350 亿美元一度上升至 1.95 万亿美元，涨幅接近 500%。管理的基金数量也由 2000 年的 2840 只上涨接近 3.5 倍。2008 年金融危机期间，受业绩表现不佳且投资者大量赎回的影响，到 2009 年 4 月，全球对冲基金规模缩减至 1.29 万亿美元。2009 年之后，在全球经济复苏背景下，对冲基金规模又开始反弹，至 2013 年 11 月底，全球对冲基金共管理着 1.99 万亿美元的资产，对冲基金规模重回增长轨道，并于 2015 年创出历史新高。

## 1.4 中国量化投资的发展历史

国内量化交易起步较晚，真正兴起是在美国次贷危机之后，大批华尔街资深人士回国发展，根据量化交易策略使用程度的不同，并且结合第一财经《宽客在中国》系列报道的资料，量化交易的本土化发展大致也可以分为 3 个阶段，分别是 ETF 套利时代（2010 年以前）、多因子 Alpha 和高频交易称雄时代（2010—2015 年），以及多元化投资时代（2016 年至今）。

### 1.4.1 ETF 套利时代（2010 年以前）

公募领域的第一只量化基金于 2002 年诞生。早期量化基金以指数增强型为主，且形单影只受冷落，从 2006 年开始的两年间无一发行。由于当时 A 股市场持续低迷，机构在量化研究上的投入也捉襟见肘。当时在量化研究方面做得还不错的某家机构团队一度濒临解散，成员四处求职，且也没什么机构愿意接纳。

直到 2009 年才开始出现加速迹象，其契机是美国次贷危机后，华尔街的量化投资人才相继回国，回国以后主要加入公募基金，富国沪深 300 指数增强基金等采用量化方法进行投资的产品也相继推出。随后，多因子选股这个概念就逐渐开始流行，可是真正做量化对冲的开始时间却是 2010 年。据《第一财经日报》统计，截至 2009 年共有 13 家基金公司推出了 15 只主动管理型量化基金，总规模约 241.04 亿元人民币，仅占公募基金总规模的 1% 左右。但值得注意的是，这些基金业绩排名都相对靠后。

尽管是公募基金最先拉开中国量化投资的序幕，但是在 2010 年以前，ETF 套利可以说是国内量化投资的主流品种。2008 年各类套利产品瞬间开始增加，整年有 10 只产品成立，套利策略涵盖 ETF 跨市场套利、可转债套利、基金封转开，以及正股+认沽权证套利等。2009 年，部分私募在 ETF 套利的基础上又开发了 ETF 延时、多头事件套利等。但随着进入的淘金者越来越多，套利空间日渐狭窄，ETF 套利市场已趋于饱和。

在国内期货领域，最早的量化投资者主要是一批经常做内外盘对冲的现货商，如伦敦铜和国内铜之间的内外盘套利交易。当时只有商品期货，所以期货市场的关注度较低，总体上算是小众市场。此外由于许多品种定价权受制于国外，隔夜的跳空较为常见，所以量化投资存在较大的风险。

## 1.4.2 多因子 Alpha 和高频交易称雄时代（2010—2015 年）

### 1. 股票市场

2010 年 4 月，沪深 300 股指期货上市，标志着股票 Alpha 策略和股指期货套利策略的兴起，但是整体来看，2010 年并未看到以股票 Alpha 策略为主的对冲产品，也未有明确标识以股指期货套利的产品。2010 年到 2012 年是一个观望期，因为很多人也不太清楚股指期货到底怎么用，大概从 2012 年年底到 2013 年年初这段时间开始，特别是在 2014 年有了一个爆发期，大家看明白了，可以利用它去选股，选出一篮子带有超额收益的股票，同时做空股指期货，进而可以实实在在地把超额收益转化成绝对收益。因此 2010 年到 2015 年 9 月股指期货受限之前的这段时间就是市场中性策略的大爆发的时期。

2012 年量化对冲产品数量继续急剧增加，全年共发行信托产品 78 只，资管产品 17 只，共计 95 只。量化对冲策略也更趋丰富化，加入了定向增发等策略。

2013 年新增加的量化对冲基金多以股票 Alpha 基金为主，2013 年创业板的牛市让这些 Alpha 基金大赚不少，但也隐藏了一些问题，比如同质化和权重依赖创业板。这些问题在 2014 年创业板不再坚挺的时候就逐步暴露了。此时，市场中性



策略的价值就更多地体现出来了。

2014 年，基金业协会推行私募基金管理人和产品的登记备案制，推动了私募基金的全面阳光化，加速了私募基金产品的发行，其中自然也包括量化对冲型私募产品。粗略估算，2014 年全年各种方式累计发行的量化对冲产品数量超过 600 只。

## 2. 期货市场

自 2010 年股指期货诞生以来，期货量化迎来了大发展！在股指期货推出以后，期货市场受到了社会的广泛关注。而期货市场的整体成交量也出现了大幅度的增长，多个品种成交金额居于世界前列，这为量化投资奠定了良好的基础。

与此同时，期货行业也有了不小变化。首先，期货公司纷纷成立了金融工程部或者投资咨询部，证券研究所与期货研究所联系更为紧密，更注重金融产品的研究；客户对于程序化交易开始逐渐认同；市场上各种量化投资平台蜂拥而出。

一般来说，期货相对于股票债券更适合量化交易。期货市场主力品种投机性强、流动性好，很多期货品种都会同时出现双主力合约，便于进行跨期套利。不同于股票 T+1 制度，期货采用 T+0 制度，所以期货投资者更加青睐于短线交易。同时，相对于股票动辄要上千万资金的程序化接口，期货程序化交易接口资金门槛低得多，更有利于实现量化交易。而且期货成交量与持仓量的高倍数也使得期货投机氛围更浓厚，价格波动比股票大，所以高收益产品更受青睐。

## 3. 高频交易

高频交易是一种对电脑接收金融指令的速度、交易的反应处理速度、发出指令的速度要求都极高的交易方式，许多交易商能在短短的一秒钟内发出数千个交易指令，并随后在几毫秒内根据最新的市场信息和模型的参数立即取消或转换指令。目前，美国 2% 的高频交易商的交易量却占到了股票市场总成交量的 60%~70%。与美国不同，目前国内市场上已有快速交易和程序化炒单，但并不存在类似国外饱受争议的“闪电交易”，由于国内交易所发布数据的频率通常为每





秒钟 2 笔或 4 笔，交易者无论利用什么软件，行情发送频率都是一定的，无法实现毫秒级别的快速开平仓，因此多数交易者都会持仓数秒钟以上，与国外所谓的高频交易有着本质区别。

随着 2010 年股指期货上市，国内高频交易迎来了春天。这群低调的人以毫秒为时间维度快速下单，每日交易量达数千手甚至更多，每月 90% 以上交易日收益为正。至此股指期货交易规模急速膨胀，仅 2014 年一年的时间，沪深 300 股指期货的交易额就达到 163 万亿元人民币，2015 年 6 月，股指期货日均成交量达到 2.3 万亿元人民币的峰值，股指期货主力合约的成交量约为 240 万手。

### 1.4.3 多元化投资时代（2016 年至今）

2015 年 9 月，中金所对股指期货实施史上最严厉监管，提高 10 倍手续费，股指期货交易量骤降为 180 万手，对于手续费极度敏感的高频做市商没法做了。但对于日内趋势商，平均单笔净利润有 1 万元人民币左右，手续费从 150 元人民币提高至 1500 元人民币，伤害不大，因此依然可以继续交易，市场波动有增不减，接着交易所将日内平仓手续费提高至 100 倍，一时间交易量下降 99%，股指期货市场基本上消停了。

股指没法做了，高频交易只好转战商品期货，2015 年下半年是商品的大熊市，无数商品跌至低点，价格便宜，手续费自然就低了，高频做市商大有可为。2016 年商品大牛市，焦炭翻了三四倍，趋势明显，商品日内、商品趋势高频交易又活跃了起来。可惜好景不长，交易所连续提高焦炭等品种手续费率，几乎每天翻一倍，抑制“黑色系”商品过度投机。高频交易进入举步维艰时代。

期货市场流动性趋于枯竭，在此情境下，市场上已经聚集起来的量化团队开始逐步转型：一方面，从低收益、低风险的套利对冲策略，逐步向多空策略、股票多头策略转变；另一方面，从股指期货向商品期货、国债期货等品种的 CTA 策略转变，看似被动，实则开辟了量化投资的新时代。

此时，CTA 基金开始进入人们视野。CTA 策略其实很早就有了，2010 年到





2015 年间，股票对冲风头正旺，CTA 只是零头，大公司看不上，2015 年之后大家都重点发展 CTA 策略，据民生证券的《CTA 策略巡礼：从趋势到套利》统计，我国自 2013 年发行第一只 CTA 基金以来，CTA 基金规模迅猛发展，2016 年 CTA 基金发行量达 215 只，截至 2017 年 2 月 22 日，我国已发行 307 只 CTA 基金。

商品属于非传统资产类投资，所以基于商品的策略也被叫作另类策略。所谓的另类策略，就是回报收益比与股市的不一样，所以有利于分散投资。从美国的研究实证上来看，从 1990 年到 2013 年，标普 500 指数与 CTA 指数的相关性为-0.10 左右，这表明 CTA 策略是独立于股市的，商品类资产与股票类资产可以形成互补。所以，在 A 股方向还未明确的时候，配置 CTA 基金可作为分散投资。

## 1.5 国内常用的量化交易策略

本节简单地介绍国内常用的量化交易策略，分别是期货 CTA 策略、股票 Alpha 策略、期权波动率套利策略，以及高频交易策略。

### 1.5.1 期货 CTA 策略

商品交易顾问（Commodity Trading Advisor，简称 CTA）是指通过为客户提供期货、期权方面的交易建议，或者通过受管理的期货账户参与实际交易，来获得收益的机构或个人。1949 年，美国证券经纪人理查德·唐川（Richard Donchuan）设立第一个公开发售的期货基金，标志着 CTA 基金的诞生。1971 年，管理期货行业协会（Managed Future Association）的建立，标志着 CTA 正式成为业界所接受的一种投资策略。传统意义上，CTA 基金的投资品种仅限于商品期货，但近年已扩展到包括利率期货、股指期货、外汇期货在内的几乎所有的期货品种。CTA 基金因为很好的业绩稳定性和与其他策略的低相关性，赢得了快速的发展。

广义上，CTA 策略基本上能够分三大类，其中，趋势跟踪策略约占 70%、均值回归（有时也叫价差套利）占 25%左右、逆势或趋势反转占 5%左右。但是

由于趋势跟踪策略所占比重巨大及国内习惯把趋势跟踪策略等同于 CTA 策略（狭义理解），故在以后的章节中讲述的 CTA 策略特指趋势跟踪策略。

### 1. 趋势跟踪策略

趋势跟踪策略是基于市场并非有效的假设，基本面变化的信息需要一定的传递时间，资产价格不能立即反映基本面的变化，价格向合理方向逐渐变化的过程所表现出的趋势。与正态分布相比，资产收益率的分布通常具有“尖峰肥尾”的特点，“肥尾”提供了趋势跟踪策略的收益。趋势跟踪策略的赢利与市场的波动性密切相关，存在亏损的可能，因此，趋势跟踪策略交易者要注意及时止损。趋势跟踪策略在一定程度就是“追涨杀跌”的策略，通过快速止损实现“大赢小亏”，从而在整体上获利。一般情况下，国内习惯把趋势跟踪策略简称为 CTA 策略。

该策略是以某资产价格的历史信息为基础，要么设置一个价格正常波动的范围（即通道），当价格突破这个范围时采取策略；要么通过长短期均线的相对运动趋势采取策略。这类策略在本质上都是一种基于市场波动强度的投资策略，在市场波动剧烈时容易获利，在市场波动较小时，收益率较低。并非每次交易都能获利，及时止损从而实现“大赢小亏”，才是趋势跟踪策略的目标。

根据所跟踪趋势级别的不同，趋势跟踪策略可分为日内短线策略和日间中长线策略。

#### （1）日内短线策略

日内短线策略要求所有开仓头寸都必须在日内交易时段结束前平仓出局，这种策略下资金暴露在风险中的时间最短，能获得稳定的利润收益，但也要求所选择的投资品种必须在日内有较大的波动和成交量，故这种策略选择的投资品种多为豆粕、螺纹钢、橡胶等商品。在本书之后的章节会详细地介绍由 vn.py 官方实现的几种经典日内短线策略。

#### （2）日间中长线策略

日间中长线策略主要把握市场或标的品种的中长期趋势，不同于短线策略主



要依赖短期技术模型的胜率优势，建立在中长期趋势至上的量化交易策略，不仅关注技术分析，而且设定一套针对期货或其他金融品种基本面分析的基本面模型，或者两者结合。当然，这种量化交易策略的实际获利，也不会像短线策略那样被快速反映在账面之上，实际量化模型的有效性和收益率情况，也往往需要更长时间的验证。

## 2. 价差套利策略

价差套利策略，通过捕捉市场的不合理价差，买入被低估的资产，卖出被高估的资产，获得回归收益，达到资本赢利或避险的目的。套利交易风险小、回报稳定，对于大资金而言，如果单边重仓介入，将面临持仓成本较高、风险较大的不足；反之，如果单边轻仓介入，虽然可能降低风险，但其机会成本、时间成本也较高。因此整体而言，大资金单边重仓或单边轻仓介入期市，均难以获得较为稳定和理想的回报。而大资金如以多空双向持仓介入期市，也就是进行套利交易，则既可回避单边持仓所面临的风险，又可能获取较为稳定的回报。

价差套利进一步细分为跨期套利、期现套利、跨品种套利、跨市场套利这四种。

### （1）跨期套利

跨期套利是指在同一市场上同时买入、卖出同种商品不同交割日的期货合约，以期在有利时机同时将这两个交割月份不同的合约对冲平仓而获利。跨期套利是套利交易策略中最普遍的一种，可以通过对冲和交割两种方式平仓。导致配对资产价格差的主要原因是资金的不均衡和季节性因素，两合约上资金的不平衡，使得某个合约的波动速度要明显快于其他合约，从而出现套利机会。跨期套利在同一交易所内完成配对资产的交易，不需要划转资金，容易实现账面平衡。

通常情况下，跨期套利只发生在期货价格大于现货价格的情形下，因为期货价格小于现货价格时，相应操作属于投机而不是套利。以对冲进行套利时，若市场处于牛市，会导致近月合约价格上升幅度大于远月，或近月合约价格下降小于远月，此时应“买近卖远”；若市场处于熊市，会导致近月合约价格上升幅度小于远月，或下降幅度大于远月，此时应“卖近买远”。

跨期套利主要涉及季节性波动套利，而季节性波动主要是由供需的季节性变化导致的。只要供需结构不发生较大的变化，季节性波动套利的模式就有可操作性。具体到期货品种，可以考虑螺纹钢、铁矿石、焦煤、焦炭、鸡蛋、豆粕、棕榈油、白糖、塑料、玻璃和沥青期货。

季节性波动套利的焦点在于不同月份合约的强弱变化，关注的合约组合是 1 月和 5 月组合，以及 9 月和 1 月组合。以塑料为例，每年的 2 月到 3 月为春季地膜的消费旺季，9 月到 10 月为秋季地膜的消费旺季，1 月与 5 月合约价差在四季度走强，9 月与 1 月合约价差在三季度走强，因此可以在地膜消费旺季到来之前开始布局价差套利策略。

## （2）期现套利

期现套利是指某种商品期货合约，当期货市场与现货市场在价格上出现差距，从而利用两个市场的价格差距，低买高卖而获利，如图 1-1 所示。理论上，期货价格是商品未来的价格，现货价格是商品目前的价格，按照经济学上的同一价格理论，两者间的差距，即“基差”（基差=现货价格-期货价格）应该等于该商品的持有成本。一旦基差与持有成本偏离较大，就出现了期现套利的机会。其中，期货价格要高出现货价格，并且超过用于交割的各项成本，如运输成本、质检成本、仓储成本、开具发票所增加的成本等。期现套利主要包括正向买进期现套利和反向买进期现套利两种。

当期货价格大于现货价格时，称为正向市场。当期货价格对现货价格的升水大于持有成本时，套利者可以实施正向买进期现套利。即买入（持有）现货的同时卖出同等数量的期货，等待期现价差收敛时平掉套利头寸或通过交割结束套利。

当期货价格小于现货价格时，称为反向市场。反向套利是构建现货空头和期货多头的套利行为（在期现套利中就是做空基差），由于现货市场上不存在做空机制，反向套利的实施会受到极大的限制。



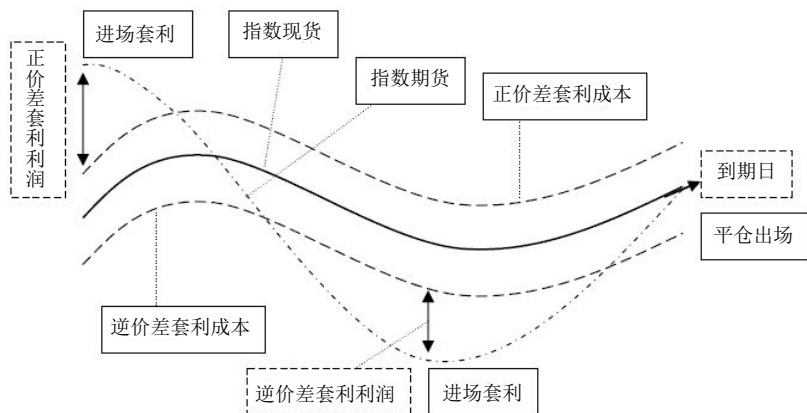


图 1-1 期限套利

### (3) 跨品种套利

跨品种套利是利用存在相关性的两种商品的期货合约价格差进行套利交易，即买入某一交割时间某种商品的期货合约，同时卖出另一相同交割时间、相关联的商品的期货合约，以期在合适时机将这两种合约同时对冲平仓从而获取利润。跨品种套利的本质是寻找价格差具有相对稳定关系的相关性的商品，并捕捉两者价格差偏离正常状态的情形，采取相关的反向操作获取利润。

跨品种套利的品种一般有两类：一是选择产品与原材料，二是选择能互相替代的产品。具体国内市场而言，跨品种套利一般可以在以下品种中进行。

- 螺纹钢与铁矿石、焦炭

钢铁生产中最重要原料就是铁矿石，其次是焦炭。钢铁生产的技术流程现已十分成熟，没有大的变化。生产 1 吨生铁，大约需要 1.5~2 吨的铁矿石、0.4~0.6 吨的焦炭。因此，钢铁的价格基本上取决于铁矿石与焦炭的价格。钢铁与铁矿石的相关性很强，与焦炭的相关性次之。

- 大豆与豆油、豆粕

豆油是常用的食用油，而豆粕则可以作为动物饲料。压榨加工大豆，可以产

出豆油并剩下豆粕，因此这三者之间可以进行跨品种套利。一般而言， $100\% \text{大豆} = 18.5\% \text{豆油} + 80\% \text{豆粕} + 1.5\% \text{消耗}$ 。

- 焦煤与焦炭

焦煤是焦炭的上游产业，按照现在的生产技术，1.3 吨焦煤可以产出 1 吨焦炭。因此，二者价格相关性强，可以进行跨品种套利。

- 热轧卷板与螺纹钢

热轧卷板是一种钢板，以板坯为原料，加热之后进行粗轧和精轧后产出。热轧卷板作为一种重要的钢材，广泛应用于基建、船舶、汽车等领域。

热轧卷板与螺纹钢同为钢材，原材料成本相近，因此两者价格具有较好的相关性。然而，由于下游消费市场具有差异，两者短期的供需关系会有不同，也就有了套利机会。

- 豆油、棕榈油与菜籽油

豆油、棕榈油与菜籽油均为食品添加剂，互为替代品。一般情况下，豆油与棕榈油、豆油与菜籽油的相关性较强，而棕榈油与菜籽油的相关性则相对弱些，因此推荐使用豆油与其他两个品种进行套利。

豆油的原料大豆主要产自美国、巴西及阿根廷，而棕榈油则一般产自印度尼西亚和马来西亚。由于不同地区的气候差异等因素，豆油与棕榈油的价差往往会出现波动，为投资者提供了套利机会。

由于菜籽油营养更为丰富且原料价格高，菜籽油的价格一般高于豆油，两者的价差一般较为稳定。同样，价差受到季节性气候等的影响，会出现一些跨品种套利机会。

- 强麦与玉米

强麦指强筋小麦。小麦和玉米是世界范围内重要的两种农作物，在粮食和饲



料市场中占据相当大的份额。两者互为替代品，价格具有同涨同跌的大趋势。但由于两者的收获季节不同，受气候等因素的影响也不同，因此价差会出现波动，提供跨品种套利机会

- 沪深 300 指数与上证 50 指数、中证 500 指数

由于沪深 300 指数、上证 50 指数、中证 500 指数成分股之间存在差异，所表现出的市场走势特征有所不同。根据 2007 年以来市场实际走势来看，整体而言沪深 300 指数及上证 50 指数走势的相关性极强，无论是长周期或短周期，在上涨阶段或下行阶段，或震荡阶段，沪深 300 指数及上证 50 指数走势的相关性均在 95% 以上。而中证 500 指数与其他两大指数的相关性较弱，由于成份股的差异性，中证 500 及上证 50 指数的相关性最弱。

据兴证期货研发中心统计，沪深 300 指数及中证 500 指数的相关性在市场处于上涨阶段时相对较强，相关系数至 78%；在市场处于下跌阶段时，中证 500 指数及其他两大指数的相关性较弱；在市场处于震荡阶段时，沪深 300 及上证 50 指数的相关性更强，走势相关性达 99% 以上。

#### （4）跨市场套利

跨市场套利即对同一期货品种在不同市场间进行套利。国内 3 个商品期货交易所并没有重复的品种，因此跨市场套利一般在国内和海外的期货交易所之间进行。对于同一种商品，交易所与原产地的距离会影响价格。

对于国内投资者而言，主要有以下几个海外市场可供套利。

- 芝加哥期货交易所（CBOT）

芝加哥是美国最大的谷物集散地，而芝加哥期货交易所早期也已有农产品的交易，如大豆、玉米、小麦。经过漫长的发展，现在的交易系统已经非常稳定和成熟。因此，我国大连商品交易所（简称大商所）的大豆、玉米及郑州商品交易所（简称郑商所）的强麦，均可与其进行跨市场套利。



- 伦敦金属交易所（LME）

伦敦金属交易所成立于 1876 年，是世界上最大的有色金属交易所。伦敦金属交易所采用国际会员资格制，绝大多数的交易来自海外市场。交易所的交易品种有铜、铝、锌、铅等有色金属，可以与上期所相应的金属期货进行跨市场套利。

- 马来西亚衍生品交易所（BMD）

马来西亚衍生品交易所所有世界上最具流动性和运作最成功的毛棕榈油期货（FCPO）合约，可与我国大商所的棕榈油期货进行跨市场套利。

- 纽约商品交易所

纽约商品交易所分为 NYMEX 和 COMEX 两个部分，其中 NYMEX 主要进行能源类商品的交易，而 COMEX 主要进行金属类商品的交易。COMEX 具有全球最大的黄金期货交易市场，同时也有银、铜、铝等期货和期权合约。我国上海期货交易所（简称上期所）的多个金属类期货可以与其进行跨市场套利。

- 东京工业品交易所（TOCOM）

东京工业品交易所成立于 1984 年 11 月 1 日，是一家综合商品交易所，曾经是世界上最大的橡胶交易所。橡胶期货合约（RSS）于 1952 年 12 月 12 日上市交易，是世界上最早的天然橡胶期货合约。日本作为橡胶的消费国，RSS 合约至今仍有足够的成交量。因此，可与我国上期所的橡胶期货进行跨市场套利。

### 3. 反趋势策略

趋势跟踪策略追踪趋势，反趋势策略则预测拐点。反趋势策略通常运用头肩形态、突破形态、交易量等反转指标来发现趋势的转折信号，建立头寸。但是在实际运用中，反趋势策略非常小众。

不管怎样，反趋势策略提供一种与趋势跟踪策略同样有效的系统性的、保守的交易框架，使用的却是完全相反的方法。与趋势跟踪策略相比，反趋势策略通常交易区间更短，成功率更高一些，成功/失败比率更小一些。一个典型的反趋势





策略将会比趋势跟踪策略交易更频繁一些，成功率在 55%~60% 之间，成功交易与失败交易的比率小于 1.5。

大部分反趋势模型寻找卖掉短期内超买的和买入短期内超卖的机会。这有点像在等待橡皮筋拉伸到它的极限时机，打赌它们会回弹到一个相对松弛的状态。这些行为使得反趋势交易模型在市场缺乏方向性或者波动性很大时斩获颇丰，并能够在市场拐点出现的时候迅速反应。反趋势模型的缺点是在稳定的、趋势性较强的市场环境中交易经常不顺，也就是常说的“赢小钱亏大钱”。

一个短期的反趋势模型想要成功，价格必须要在指数的长期趋势或者某些其他的价格点附近不断摇摆。这种市场反应意味着市场价格移动包含足够的噪声及波动性，从而带来反趋势交易的赢利机会。

## 1.5.2 股票 Alpha 策略

股票对冲策略通过做多/做空两种方式来投资股票及其衍生品（如股指期货、融资融券等）。投资范围可以是全市场，也可以专注于某些特定行业、主题。不同的基金在净风险敞口、融资杠杆、持有期、持股集中和持有股票价格范围方面有很大的差异。国内比较常见的是 Alpha 策略，即运用复杂的量化方法从技术面、基本面角度分析未来价格变动趋势，以及不同股票间的相关性，进而买入低估值股票的同时卖出高估值股票，或者通过股指期货对现有投资组合头寸进行完全（或部分）对冲，隔离系统风险，获取 Alpha 收益。该策略的成功取决于量化选股模型的有效性、对冲的覆盖程度，选股模型越有效，系统风险对冲得越好，策略效果越好。

### 1. Alpha 的含义

拓展的资产定价模型（CAPM）如下所示。

$$R_s = a + R_f + \beta_s(R_m - R_f) + \epsilon_s$$

其中， $R_s$  为现货组合的预期收益率， $R_f$  为无风险利率， $R_m$  为市场指数的预期

收益,  $\epsilon_s$  为误差项,  $\alpha$  衡量了非系统性风险,  $\beta_s$  衡量了系统性风险。投资者在市场中同时面临着系统性和非系统性风险, Alpha 策略通过对系统性风险进行度量并将其分离, 从而获取超额绝对收益。

### (1) Alpha 和 Beta 的关系

市场上常见的指数基金表现为: 如果整个市场涨了, 业绩也跟着涨, 但如果整个市场跌了, 业绩也跟着跌。因为它的 Beta 系数一般在 1 左右, 它的收益主要来源于 Beta。

若把投资组合收益率分解成 Alpha 和 Beta 两部分以后, 发现一个最重要的事实, 这两部分的价值是不一样的。简单地说, Alpha 很难得, Beta 很容易。只要通过调节投资组合中的现金和股票指数基金 (或者股指期货) 的比率, 就可以很容易地改变 Beta 系数, 即投资组合中来自整个市场的收益。

因此 Beta 很便宜, Alpha 却很贵。指数基金和 ETF 基金是购买纯 Beta 的工具。因为只有 Beta, 所以它们一般只收取很低的管理费。没有 Alpha, 所以它们一定不会收取基于利润的分成费。在主动型公募基金, 基金经理试图获得更好的绩效, 也就是除 Beta 以外还想得到更多的 Alpha。想获得 Alpha 靠的是真本领。Beta 只是随大势, 但“水可载舟, 亦可覆舟”。国内的许多基金都只有 Beta, 当然这在很大程度上是因为缺乏金融工具的选择, 比如在融资融券出台之前不可以沽空。大盘开始暴跌的时候, 也就是“股神”神话破灭的时候。业内人士有个比喻: Alpha 是肉, Beta 是面。指数基金全是 Beta, 卖的是馒头; 主动型公募基金卖的有肉有面, 是包子; 而对冲基金卖的就是纯肉。肉比包子贵, 包子比馒头贵, 贵表现在其收费模式是“2-20”, 即 2% 基础管理费和 20% 业绩提成。

### (2) Alpha 策略的基本思想

Alpha 策略是典型的对冲策略, 通过构建相对价值策略来超越指数, 然后通过股指期货或期权等风险管理工具来对冲系统性风险。Alpha 策略属于市场中性策略, 但是 Alpha 策略的约束更小, 其 Alpha 来源可能是行业的、风格的或者其他的。Alpha 策略注重选股, 属于主动投资, 相比之下, Beta 策略注重对投资时



机的选择，属于被动投资。

虽然量化策略的最终目标是通过各种手段获取稳定的 Alpha，但是由于当今市场政策，完全对冲 Beta 成本过大，而且期权标的物都是大盘股，不能代表整个市场。以上种种原因导致产品无法进行完全对冲，所以现实中多数产品的收益是由 Alpha 与 Beta 两部分组成的。Beta 提供的收益来源于产品对市场及风格（大小盘、成长价值等）的敞口，Alpha 提供的收益来源于产品管理人的管理能力，是产品收益剔除了 Beta 带来的收益后所剩余的部分。所以很多打着“Alpha 策略”标志的产品最终的业绩表现实际上是由 Alpha 与 Beta 共同决定的。通过量化的手段，产品仅能够获取稳定的 Alpha，但是 Beta 带来的收益并不稳定。只有在产品 Beta 敞口与市场偏好相符时，Beta 才会进一步增强产品的收益。反之，Beta 敞口会明显削弱产品的收益。考虑到 Beta 对于收益的影响强于 Alpha，市场整体表现的变化或者市场风格的切换都可能对于产品最终的业绩表现产生明显的影响。

### （3）Alpha 策略的分类

在实际中经常使用的 Alpha 策略主要有多因子、风格轮动、行业轮动、资金流、动量反转等。

多因子是应用最为广泛的一种策略，该策略选择一系列因子来搭建模型。通过这些因子筛选股票，满足则买入，不满足则卖出。多因子的最大优势在于，在不同的市场和行情下，因子库中总有一些因子能够发挥作用。

风格轮动是指利用市场的风格特征进行投资。市场有时会偏好小盘股，有时偏好大盘股。通过观察某些指标来判断市场的倾向性，在风格转换的初期介入，可获得较大的超额收益。

行业轮动是指市场在经济周期的作用下对各个行业产生不同的偏好。在经济周期中，我们可以按照顺序依次对各个行业进行资产配置，从而获取相比于“买入-持有”策略的超额收益。

资金流是根据资金的流向来进行选股。对于一只股票，资金流入时，股票的

价格应该会上涨；资金流出时，股票的价格应该会上跌。通过观察资金流的情况，我们可以预测未来股价的变化。

动量反转是指股票的强弱变化情况，过去一段时间强的股票，在未来一段时间继续保持强势，过去一段时间弱的股票，在未来一段时间继续弱势，这叫作动量效应。过去一段时间强的股票在未来一段时间走弱，或者过去一段时间在弱的股票在未来一段时间走强，这叫作反转效应。通过判断动量反转的有效性，筛选出应该购买的股票。

#### （4）Alpha 策略的优势

Alpha 策略有三大优势：一是回避了择时这一难题，仅需专注于选股；二是波动较单边买入持有策略要小；三是在单边下跌的市场下也能赢利，Alpha 与市场的相关性理论值为 0。在熊市或者盘整期，可以采用“现货多头 + 期货空头”的方法，一方面建立能够获取超额收益的投资组合的多头头寸，另一方面建立股指期货的空头头寸以对冲现货组合的系统风险，从而获取正的绝对收益。

## 2. 因子的分类

下面开始介绍最常见的 Alpha 策略：多因子策略。

对于因子的分类方法很多，整体而言，因子可以被分为基本面因子和技术面因子。基于对一只股票的不同特征的刻画，一般而言，可以将因子更加细致地分为：赢利性、估值、现金流、成长性、资产配置、价格动量和技术面因子。

#### （1）赢利性

与赢利性相关的因子主要反映了公司利用现有资源实现收益的能力。公司的赢利能力可以通过许多方法来衡量，例如投入资本回报率（ROIC）、已利用资本回报率（ROCE）、净资产收益率（ROE）、总资产收益率（ROA）、边际利润、人均收入、经济利润、投资增额收益率。整体而言，赢利性因子是一类效果较好的因子，即赢利性好的公司股票具有显著的正超额收益，而赢利性差的公司股票具有显著的负超额收益。



## （2）估值

估值因子主要反映了股票作为一种资产的价值与其价格的相关性，但其决定性因素是该公司为其客户创造价值的多少。估值可以通过许多方法得到，但都包括了一定的定性分析和对未来的预测。常见的估值因子有：自由现金流价格比、外部融资总资产比、企业价值与 EBITDA 比（EV/EBITDA）、市盈率、股息率等。

市销率可以说是美国股市最有效的因子，但是在中国股市却失效。国内分析师团体更倾向于用市盈率来进行估值，主流研报上市盈率也更有市场，故市盈率可以说是中国最有效的估值因子。

## （3）现金流

现金流可以分为经营性、投资性和融资活动三类。其中，经营性现金流，包括从商品销售和服务得到的现金减去生产这些产品和提供这些服务需要支付的现金流出，包括为利润支付的现金税和为债务融资支付的利息。一个公司产生的经营性现金流水平是衡量未来股市回报的一个重要指标。常见的现金流因子有：自由现金流和营业收入之比、投入资本现金回报率等。

## （4）成长性

成长性因子在市场中通常获得的超额收益较为微弱。这主要因为成长性投资更多是定性而非定量的，更加依赖投资者独到且有前瞻性的眼光而非精确的数据分析，更加偏向于“艺术”而非“量化”。尽管如此，成长性仍然是我们因子库中重要的一部分。正如成长性投资者们所说的：“我所知道的投资中最大的一个错误，就是对那些最伟大公司和其他普通公司一视同仁。”在实际使用成长性因子的过程中，我们常常和其他因子结合使用，以弥补其预测性不足的劣势。常见的成长性因子有：每股自由现金流、每股盈余等。

## （5）资产配置

资产配置主要涉及一家公司资本资源的使用情况，主要考虑现金来源和现金使用两方面的内容。现金来源主要包括业务经营、资产和投融资收入、发行股票

和发行债券等。现金使用主要包括经营费用、业务投资、业务收购、项目或证券投资、支付现金股利、偿还债务及回购股份等。常见的资产配置因子有：净回购股份与投入资本比、一年流通股减少量、一年长期债务减少量、外部融资和总资产比、三年平均资本支出和投入资本比、收购与投入资产比等。

### （6）价格动量

价格动量因子能够在一定意义上反映市场周期和投资者情绪，并依此对未来进行预测。衡量价格动量的一般指标是价格变化的速度，或一段时间内价格的变化率。正的价格动量意味着某只股票的买家数量正在不断增加，需求大于供给；负的价格动量则意味着供需平衡向卖家倾斜，供给大于需求。常见的价格动量因子有：相对强弱、价格范围、相对强弱指数等。

### （7）技术面因子

技术面因子相比于基本面因子，数据更新时间快，更加注重市场的微观结构，而非股票的价值。常见的技术面因子有：强弱指标（RSI）、随机指标（KD）、趋向指标（DMI）、平滑异同平均线（MACD）、能量潮（OBV）等。由于技术面因子的 Alpha 往往不稳定，所以在实际应用中较为少见。

## 3. 因子的筛选和评价

因子筛选的前提是获取充足的历史数据，包括基本的股价历史行情、基本面数据、分析师情绪指数、宏观经济数据等，可用 Tushare 获取各种免费或收费的历史数据，对数据质量要求更高的可用 Wind 或者 Bloomberg 获取。通过对这些海量数据进行分析，力求从中找出具有显著超额收益的因子。

### （1）因子筛选的整体思路

上市公司的数据多而复杂，在对这些数据进行处理之前，我们需要对数据进行选择，即对因子进行初步的筛选。筛选因子的主要原则有：

- 数据的准确性和真实性；



- 数据的完整性;
- 数据来源的稳定性。

## (2) 因子评价的整体思路

在完成了因子的初步筛选之后,需要对因子进行进一步的评价。因子评价的整体思路是研究各个股票的超额收益和因子参数之间的关系。我们试图找出这样的因子:对于绝大多数股票而言,当该因子参数越大/越小时,超额收益越大/越小,或者恰好相反。总之,我们试图找出那些能够有效预测一只股票未来超额收益的因子,无论两者是正相关还是负相关。

研究股票超额收益和因子参数之间关系的方法主要有两种:

- 根据因子参数的大小对股票进行分组,计算每组的平均超额收益,并依次进行因子胜率、 $t$  检验。
- 在每一个时间点上,计算全体股票截面上的超额收益率和因子参数大小的相关系数,以及信息比率。

## 4. 因子的组合

### (1) 冗余因子的剔除

由于很多因子内在的驱动原因大致相同,所以通过这些因子选出的股票往往很相似,即统计学意义上的自相关性,因子不能相互独立。所以需要剔除掉一些有效但是信息冗余的因子,即在同类的因子中只需要保留收益最好、区分度最高的那一个。

剔除冗余因子的一般方法如下:

- 取出各个有效因子在各个时间点上关于分组的序列;
- 计算这些序列的相关性矩阵;
- 得到相关性矩阵的时间序列,并求该时间序列的均值;
- 通过这个均值矩阵挑出相关性较大的因子组;



- 对于每个因子组，挑选其中有效性最好或者收益最好的一个因子作为最终的因子。

## （2）因子的降维

在多因子模型的实际应用中，希望将有效的因子加以组合和处理，减少模型中变量的个数，这种减少自变量的过程叫作降维。降维有利于防止过拟合，过拟合就是把历史数据的噪声也拟合过来了，所以因子并不是越多越好。

像牛顿第二定律  $F=ma$ ，仅仅 2 个因子就解析万物的运动规律；爱因斯坦的质能方程  $E=mc^2$ ，尽管推导的过程非常复杂，用到的数学工具非常高深，但最终结论却如此之简洁漂亮；“股神”巴菲特的投资方法也被后人总结成仅仅靠 6 个因子就可以战胜市场。

降维的主要方法有：因子简单平均降维法、因子历史平均收益率加权平均降维法、逐步回归分析、主成分分析等。

- **因子简单平均降维法**：因子简单平均降维法就是对同类的因子进行简单的等权平均处理，对因子参数求平均，作为新的复合因子。
- **因子历史平均收益率加权平均降维法**：加权平均降维法就是对同类的因子按照历史平均收益求加权平均，因子的历史平均收益取各个时间点分组的第一组的收益。
- **逐步回归分析**：在实际的多元回归问题中，我们总试图找到所谓“最优”回归方程，主要是指希望在回归方程中包含所有对因变量  $y$  影响显著的自变量而不包含对  $y$  影响不显著的自变量的回归方程。逐步回归分析正是根据这种原则提出来的一种回归分析方法。它的主要思路是在考虑的全部自变量中按其对于  $y$  的作用大小、显著程度大小或者说贡献大小，由大到小地逐个引入回归方程，而那些对  $y$  作用不显著的变量可能始终不被引入回归方程。另外，已被引入回归方程的变量在引入新变量后也可能失去重要性，而需要从回归方程中剔除出去。引入一个变量或者从回归方程中剔除一个变量都是逐步回归的一步，每一步都要进行  $F$  检验，以保证在引入新变量前回归方程中只含





对  $y$  影响显著的变量，而不显著的变量已被剔除。

- **主成分分析**：主成分分析的基本思路是将原来具有相关性的一些指标组合成一组新的互相无关的综合指数来代替原来的指标。一般情况下，用原来指标的线性组合作为新的综合指标。我们认为一个综合指标的方差越大，其包含的信息也就越多。因此，在所有线性组合中，用方差最大的那一个作为第一主成分。如果认为第一主成分不能有效地反映原来的信息，我们就取另一个和第一主成分相关系数为 0 的线性组合作为第二主成分，依此类推。

### （3）因子权重的确定

在完成了因子的筛选和降维之后，需要确定因子权重。对因子赋权的方法有很多，在此简要介绍三种：等权赋值、回归赋值、IC-IR 因子赋值。

- **等权赋值**：等权赋值是指在组合各个因子时对各因子赋以相等的权重。
- **回归赋值**：回归赋值是指在组合各个因子时，我们对某个时间区间上的收益率和参数因子进行最小二乘法回归，回归所得的系数向量即为各个因子的权重向量。
- **IC-IR 因子赋值**：IC-IR 因子赋值是指在组合各个因子时，考虑因子的 IC 序列，优化因子组合的 IR 值，取使得 IR 值最大的组合权重为最终的权重。

## 5. 基于因子库选股

在完成了因子的筛选和组合之后，就基本建立起了自己的 Alpha 因子库。基于这个因子库，可以筛选出这些因子较为突出的股票，并通过这些股票实现因子的超额收益。常见的选股方法有两种，分别是打分法和回归法。

### （1）打分法

打分法就是根据各个因子的大小对在一定时间内（如每 2 周）对一篮子股票进行打分，按照一定的权重相加得到一个总分，通过分数的高低进行股票的筛选，如购买前 50 名股票。基于周期打分循环，每 2 周调一次仓位。打分法的特点是比较稳健，不易受到特殊值的影响。

## (2) 回归法

回归法就是用过去的股票收益率对多因子模型进行回归，得到回归方程，把最新的因子值代入回归方程中得到一个对于未来股票值的预测，根据这个预测来进行股票的筛选。回归法的优点是能够比较及时地调整股票对各个因子的敏感性，但是回归法比较容易受到极端值的影响，导致选股失败。

### 1.5.3 期权波动率套利策略

期权的套利可分为无风险套利和风险套利。

无风险套利以平价公式套利为核心，辅以贴现套利、盒式套利等，其原理主要是捕捉市场交易价格与其理论价值之间差异的交易机会，并通过行权机制予以套利空间锁定的保障。由于机构投资者的大量参与及市场交易机制效率的提升，无风险套利的应用空间已大幅缩窄，策略市场容量也相对较少。

风险套利原则上都是在试图尽可能剥离掉其他因子的影响后，对期权组合中的某一风险因子进行“低买高卖”实现获利目的，如波动率、相关性及时间价值等，因此对于风险因子高低程度及未来变化趋势判断正确与否决定了此类策略的最终损益，其中要数期权波动率套利策略最为常见。

#### 1. 期权的特性

波动率交易听起来很奇怪，它是由期权本身独特的属性导致的。首先，期权是非线性衍生工具，其价值由两部分组成：内在价值和时间价值。内在价值取决于标的物的价格相对行权价的涨跌；而时间价值与到期日密切相关，它是随着持有时间增加而衰减的。

根据 Black--Scholes 期权定价公式，期权的价值由 5 个因素组成，每一个因素的改变都会导致期权价格的改变，对期权价格影响最大的是标的价格 ( $S$ )，其次是隐含波动率 ( $\sigma$ )，下面详细介绍一下这 5 个因素。

- 标的价格 ( $S$ )：标的价格上涨会增加看涨期权价值，但它们的关系并非线性，



即标的上涨 1%，看涨期权 Call 可能会上涨 1.5%，期权上涨由 Delta 和 Gamma 两个部分组成。

- Delta 衡量标的资产价格变动时期权价格的变化幅度，当标的价格上涨  $s\%$  时，Delta 对看涨期权 Call 的贡献是增加  $s \times \text{Delta}$ 。
- Gamma 衡量的资产价格变动时 Delta 的变化幅度，期权价格变动相对于标的物价格变动的二阶导数。Delta 和 Gamma 的关系类似于速度和加速度的关系。当标的价格上涨  $s\%$  时，Gamma 对看涨期权 Call 的贡献是增加  $0.5 \times s^2 \times \text{Gamma}$ 。
- **执行价格 ( $K$ )**：执行价格上涨，会降低看涨期权 Call 的价格，即随着  $K$  的上升，Call 的类型会逐渐从实值期权 (ITM) 过渡到平值期权 (ATM)，再到虚值期权 (OTM)。需要注意的是，执行价格是合约一开始就定下来的，不存在执行价格变化的风险。
- **隐含波动率 ( $\sigma$ )**：标的价格波动增大会增加看涨期权的价值，因为波动率越大，意味着标的物可能涨得更快，因此能获利更多。Vega 衡量标的资产价格波动率变动时，期权价格的变化幅度，当标的价格波动率上涨  $\sigma\%$  时，Vega 对看涨期权 Call 的贡献是增加  $\sigma \times \text{Vega}$ 。一般而言，远月合约相对流动性不足，敏感度更大，因此，Vega 适合远月合约操作。
- **无风险利率 ( $r$ )**：利率改变也会影响期权价格。Rho 衡量利率转变对期权价格变化幅度。由于中国是利率管制国家，所以利率变化风险可以忽略不计。
- **到期时间 ( $T-\tau$ )**：期权存在时间价值，随着到期日临近，该价值会逐渐衰减。Theta 衡量时间变化对期权理论价值的影响，表示时间每经过一天，期权价值会损失多少。

总之，期权波动率交易，本质上就是对 Delta, Gamma, Theta 和 Vega 的管理。

## 2. 交易波动率的优势

交易期权的策略主要可以分为两大类：交易标的方向和交易波动率，也就是常说的方向性交易和波动率交易。

在方向性交易中一般是不用考虑希腊值 (Delta, Gamma, Theta 和 Vega) 的, 但是会暴露更大的风险。例如买入看涨期权或看跌期权来做一个方向性的交易, Gamma 基本上没有什么用, 因为这时候你肯定已经很清楚你买入了正值的 Gamma。这个正值的 Gamma 在标的资产价格上涨时会增加看涨期权的 Delta, 在标的资产价格下跌时会增加看跌期权的 Delta。正值的 Gamma 会使得期权头寸在越来越实值的过程中 Delta 越来越大, 从而增加赢利能力。简单地说, 在方向性交易中, 正值的 Gamma 在赚钱的时候会让你加速赚钱, 在亏钱的时候会让你减速亏钱。

波动率交易对于交易方向的优势就是更低的风险和更大的收益。大量的学术研究表明, 股票的价格基本属于随机游走的状态, 波动率是标的物 (即期货) 对数收益率的方差, 而且存在均值回归, 即可以通过研究历史隐含波动率来预测未来波动率的大小。一句话总结: 标的物是涨是跌太难猜了, 波动率变化更好猜。从概率上看, 波动率交易比方向性交易的胜率更大,

### 3. 波动率套利

波动率套利的收益不依赖于标的资产的价格变动方向, 而依赖于标的资产的价格波动情况, 其核心是寻找期权的隐含波动率 and 市场的实际波动率的价差, 并对其进行相应交易。换句话说, 如果预测的波动率与期权的隐含波动率有显著不同, 就可以通过相应的期权交易进行获利。需要补充的是, 预期波动率指期权交易者根据市场情况与历史数据对未来的价格波动率做出的一种预测, 是对未来波动率的一种估量; 隐含波动率是指实际期权价格所隐含的波动率。它是利用 Black-Scholes 期权定价公式将期权实际价格以及除波动率  $\sigma$  以外的其他参数代入公式而反推出的隐含波动率。期权的实际价格是由众多期权交易者竞争而形成的, 因此, 隐含波动率代表了市场参与者对于市场未来的看法和预期, 从而被视为最接近当时的真实波动率。

在众多波动率套利策略中, 又以 Gamma Scalping 策略最为常见 (简单): 通过对波动率的预测, 每天对冲 Gamma 以获得高抛低吸的利润。



首先需要用到当月或者近月的平价期权（ATM）通过对冲 Delta 的方式构造跨式期权。平价期权拥有更大的 Gamma 值，但是近月合约由于有更好的流动性，其敏感程度相对于远月合约要小得多，即 Vega 相对较小，即 Gamma 负责赢利，Theta 负责亏损。跨式期权具有 0Delta、正 Gamma、负 Theta 和正 Vega 的特点。

若预测近期股市会出现暴涨暴跌行情，即预期波动率大于隐含波动率，则买入跨式期权。该策略的利润源自 Gamma 贡献的价值足够大来覆盖 Theta 的时间成本，也就是说  $(0.5 \times s^2 \times \text{Gamma} - \text{Theta}) > 0$ 。当行情真的有大波动时，其实现波动率要大于隐含波动率，Gamma Escaping 策略就会赢利。反之，若预测错误，Gamma 的赢利覆盖不了 Theta 的时间成本，策略就会出现亏损。

总的来说，就是：

- 预期波动率 > 隐含波动率，做多 Gamma。预测对了，当日赢利，反之亏损。
- 预期波动率 < 隐含波动率，做空 Gamma。预测对了，当日赢利，反之亏损。

#### 4. 动态对冲

与买股票价格上涨，平仓获利离场一样，当 Gamma 的贡献足够大时，也需要对冲 Delta，获利离场（假设当天建仓 Delta 为 0，Gamma 会在第二天产生新的 Delta）。相对于 50ETF 期权，对冲物有 3 种，分别是 50ETF、上证 50 股指期货和看涨看跌期权合成期货。效果最好的是用期权合成期货来对冲，其优势是杠杆高、成本低、流动性好。

有了对冲物之后，可以考虑如何进行动态对冲了。常见的有以下 3 种动态对冲方法。

（1）定时对冲：在一定时间周期内进行对冲，可以是每天收盘前对冲，也可以每隔 15 分钟或 30 分钟对冲来锁定部分日内波动。

（2）阈值对冲：通过自动对冲算法，设定 Delta 阈值，突破时瞬间对冲锁定短时间波动赢利，例如等 Delta 冲到 5000 时全部对冲，让 Delta 归零、再次积累。

(3) 智能对冲: 通过 CTA 信号或高频信号来实时判断标的物的涨跌方向, 当仍然有趋势时智能判断对冲 Delta 的数量, 这样可以捕捉到更大赢利机会, 而且节省手续费。

## 1.5.4 高频交易策略

高频交易 (High Frequency Trading, 简称 HFT) 是指从那些人们无法利用的、极为短暂的市场变化中寻求获利的自动化程序交易, 比如某种证券买入价和卖出价差价的微小变化, 或者某只股票在不同交易所之间的微小价差。这种交易的速度如此之快, 以至于有些交易机构将自己的“服务器群组”安置到了离交易所的服务器很近的地方, 以缩短交易指令通过光缆传送的时间。一般是以高频做市商/套利算法进行非常高速的证券交易, 从中赚取证券买卖价格的差价。

总的来说, HFT 可以概括为 5 方面:

- 依据市场高频数据, 使用复杂的计算机程序和算法生成订单, 并将订单送到指定的市场上去。
- 具有超低的网络信息延迟, 这通常通过“联位服务”或者“接近主机服务”将交易系统托管到交易所的数据中心实现。
- 在极短的时间内完成建仓、持仓、清仓, 通常整个过程的时间为几秒钟, 最多不超过数分钟。
- 在短时间内提交并撤销大量的订单。
- 市场中性, 不隔夜持仓。

### 1. 高频交易策略的类型

#### (1) 高频做市商

高频做市商策略是在交易所挂限价单进行双边交易以提供流动性。所谓双边交易, 是指做市商手中持有一定存货, 同时进行买和卖两方交易。这种策略的收入包括买卖价差、交易所提供的返佣和固定佣金。



每交易一笔都有返佣，返佣的数值一般很小（远远小于价格最小变动单位，比如若设价格最小变动单位为一分，那么报价只能取 100.1 元或 100.2 元），但如果交易笔数巨大，积少成多，便可以成为不菲的收入。通过赚取返佣，做市商只需要保证每笔交易不赔即可，并非一定要追求低买高卖，反而要保证自己的委托单尽可能多地被执行，以争取更大的流量。为了做到这一点，下单和改单的速度是个关键，这也是为什么这一行如今被以速度见长的高频交易商把持的原因。

佣金比返佣更吸引人。做市商只需保证每月或每天参与一定规模的交易，就可以再额外从交易所处获取一笔不小的收入。这种模式的好处在于，做市商不仅不用追求买卖价差，甚至连流量也不需争抢，只要完成限定的额度即可，难度大大降低。

## （2）高频套利

套利策略注重两种高相关性的产品之间的价差。比如说一个股指 ETF 的价格，理论上应该等于组成该 ETF 的股票价格的加权平均。但因为种种原因，有时会发现市场上这两种价格并不一致，此时即产生套利机会，可以买入价低一方，同时卖出价高一方，以赚取差价。随着市场流动性的增强，这种机会发生的次数越来越少，规模越来越小，并且机会经常转瞬即逝，因此往往需要借助高频交易的技术来加大搜寻的规模和把握交易时机。例如，其应用场景可以是跨市套利。

## （3）短趋势策略

短趋势策略即意味着预测一定时间内的价格走势。相对于低频的趋势策略，高频交易的主要数据源是比 Tick 级别数据更精确的交易委托账本（Order Book Events），所以可以在委托单的粒度上进行分析 and 预测来抓大单的动向。Tick 级别数据其实就是一种对交易委托账本的降采样，其前提假设是：最佳买卖价是最重要的信息，以丢弃其他相对不如这个重要的信息为代价，缩减数据规模，让数据处理变得更容易。



## 2. 高频交易策略的特征

### (1) 低延时

高频交易对网络延时极度敏感。解决办法是采用近邻等方法缩短交易主机与交易所之间的空间距离,通过高速或专有通信网络/网卡降低通信延时。联位服务 (Co-location) 是交易所提供的主机托管服务,把客户的主机放在交易所的数据中心,通过减少物理距离的方法以期获得订单到达交易所的最小延时。

国内四大期货交易所都有自己的机房,中金所是数讯机房,上期所在张江机房和上期大楼,大商所和郑商所在其本地也有自己的机房。只有期货公司可以租用交易所的托管机房机柜,并为量化客户提供托管服务。国内期货交易所的交易数据为每 500 毫秒一次快照,提前收到数据,可以更快做出反应,有很大优势。高频机构自购设备,自行调优后,经期货公司同意,放到租用机柜内进行交易,通过承担单独席位和柜台的成本,以达到市场最快层次的反应速度。

网络架设方案有光纤和微波。微波比光纤的延时要低很多,延时敏感的应用一定要选择这种线路。这个差距首先受制于光在光纤中的传播速度只有在空气中的  $\frac{2}{3}$  左右。另外,在大城市建筑密集地区,光纤的复杂布线会进一步增大延时,差距可能增至 2 到 3 倍。微波技术有两个主要的缺点:第一是微波在空气里传播受天气影响很大,刮风下雨都会导致通信受损;第二是带宽太小,如果是跨交易所的业务,不可能通过微波来转移大流量的市场数据,只能用来收发下单指令。所以采用微波线路时一般光纤作为备用线路。网卡要求是其网络栈上的 I/O 延时,收包、发包加起来能达到 2~3 微秒。

### (2) 专用性硬件

硬件上,新式的刀片 (Blade) 服务器被大规模地部署为高频交易主机。刀片服务器通过精简的空间设计与整合,能够大幅度地缩小主机空间,单机便能够安置 64 个 CPU,使得一个典型的数据中心能够部署上万个 CPU。这极大地降低了算法主机代管的成本,提高了近邻部署算法主机的经济性。





而专用芯片的发展进一步提升了通用性芯片的计算能力，特别是图形处理器（GPU），以及现场可编程门阵列（FPGAs）等专用芯片在高频交易领域得到重视。GPU 具有多个芯片核心和极强的运算能力，现阶段单个 GPU 具有 500 个核。通过优化程序，即可方便地利用 GPU 实现大规模运算。而 FPGAs 芯片省却了传统计算机中内存与 CPU 之间指令通信，直接实现算法与芯片之间的联系，具有极好的运算和执行优势。当前国内只允许用 FPGAs 来接收行情，用 FPGAs 来交易是违法的。

### （3）先进的算法

为了快速、实时地分析和处理海量数据，在算法设计上需要借助于一些先进、有效的方法。比如，Hadoop 即能够有效地利用多台（上万台）服务器对 PB 量级（1PB $\approx$ 1000TB $\approx$ 1 百万 GB）的金融数据进行分割处理。同时，分析计算的方法也逐渐由传统的解析统计过渡至非参统计。新式工具和方法使得高频交易的算法系统不再单一地依靠价格、交易量等信息，进一步地结合语义分析、数据挖掘等构建更为“智能”的系统。

## 1.6 宽客

宽客就是专门从事量化交易的人，在国内又称“矿工”。“宽客”这个词最早源于 2004 年出版的《宽客人生：从物理学家到数量金融大师的传奇》（*My Life as a Quant: Reflections on Physics and Finance*）。这部自传讲述的是作者伊曼纽尔·德曼（Emanuel Derman）以数学物理学者的身份转战金融领域、从高盛跳到所罗门兄弟公司的传奇故事。它描述了在一个特别的时代，科学家发现了华尔街，而华尔街也发现了科学家的过程。

书中写道，最近 40 年来，华尔街和伦敦城内绝大多数主要金融机构和很多中小金融机构中，都有一小群曾是物理学家和应用数学家的人员，尝试将物理学、数学原理应用于证券市场。以前，这些人被称为“火箭科学家”，之所以这么称

呼，是因为火箭通常被误认为是科学界内最先进的领域。现在他们通常被称为“宽客”（Quant）。宽客从事的主要工作是“数量金融”（Quantitative Finance）。这个学科是跨学科的混合体，包括物理学、数学和计算机科学等，目的是为了对金融证券进行估值。

在以前的金融学词典中曾记录着“宽客，常用作贬义词”。德曼回忆在 1985 年第一次进入高盛时，立刻为自己懂数学而感到羞愧。在挤满人的电梯里跟另一个宽客说话时，往往会以一只债券的“久期”“凸度”这样的词汇开头。这些都是一些与债券有关的数学词汇，用来描述债券价格对利率变动的敏感性。如果正在交谈的这名同事在公司待的时间比德曼长，那么他会不自在地换一个姿势，并试着转移话题，可能会模仿一个真正的债券交易员的口吻，用行话说：“期货今天可跌惨了！”很快就会明白，在一家满是交易员、销售员和银行家的公司里，如果两个聊得很投机的成年人在谈论数学、UNIX 或者 C 语言，则会被人认为这是个坏毛病，周围的人会把他们关注的目光转移。

甚至在 20 世纪 90 年代中期，宽客们还是被取笑的对象。一天下午，德曼和一个同事正站在大厅交易柜台中间狭窄的过道两边，这时一位首席交易员从他们之间走过，交易员的头正好处在他们两人头中间，突然向后退，双手紧紧夹住头仿佛承受巨大的痛苦，高声喊道：“啊！强力场！太强了！放开我！”

宽客和交易员是完全不同的两类人。交易员常以意志坚定、性格直率为荣，而宽客则更加谨慎小心、少言寡语。性格上的这种差异正是深层文化偏好上的反映。交易员受雇就是来做交易的。他们整天盯着屏幕，搜集经济信息，在电子数据表格间飞快地换来换去，运行宽客们编写的程序，输入交易信息，跟销售员和经纪人交谈，以及敲击键盘。在工作时间内宽客很难和一名交易员有广泛的交流，经常等了足足一个小时，才有 5 分钟搭话时间，而这 5 分钟还会被打断。交易员们做工作，有点像打电子游戏，结果尽管不是永远正确，但是他们学会了坚持己见，从本能出发快速思考问题，并果断做出决定。

宽客更像搞研究的学者一样，习惯从头到尾专注于一件事，而且要深入，做



得好。但在任务繁多的商业环境里，需要同时完成好几项任务，有时要停下手头还没完成的急活儿，赶紧处理另一件更紧迫的事。

宽客一开始是华尔街物种界限的违反者，他们是“混血”的选手，使得那些纯种的交易员和不含杂质的信息技术经理感到不舒服。宽客的职业定位是不清晰的业余选手，因为优秀的宽客必须是个多面手，懂交易、懂程序设计、懂数学。

21 世纪，随着高校引入金融工程的课程，金融机构逐渐重视风险管理，宽客才慢慢成为一种更加正规的职业。特别是 20 世纪 90 年代末期过热的高科技板块，让各类市场参与者都兴奋异常，同样大批对冲基金试着用数学模型从微小的美元价格变动中博取收益。

金融工程专业最初是投资银行为了满足量化岗位的需要，而与高校合作搞的短期培训班（硕士），然后慢慢扩展到本科和博士，形成相对系统的专业，现在已经变得非常热门。但是其课程的方向是对新式衍生品的定价，完全契合的工作岗位是外资投行和国内券商衍生品部的“定价 Quant”。该岗位在 2008 年次贷危机前可谓风光无限，但是随着政策变化和风控意识加强，定价 Quant 群体在逐步萎缩。而且衍生品定价的工作更偏好数学功底，故在该岗位上数学专业会比金融工程专业更有竞争力，所以金融工程专业的地位相对尴尬，它并不能立刻满足市场上对宽客的需求。

由于宽客多面手的性质，高校很难产出合格的宽客，真正的宽客一般是走出社会后自我修炼而成的。例如，金融专业毕业的人需要补数学和编程相关知识，计算机专业毕业的人要补统计学、金融的相关知识，数学专业毕业的人则要补计算机、金融的相关知识。核心是一种专研的精神，对新事物的求知欲和对该行业的热情。因此工作中觉得知识有用就可以学，爱钻研、爱较真的精神，才是宽客的立足之本。

目前市场上并没有 100% 适合宽客的证书。CFA（特许金融分析师）的内容提供了非常完整的金融知识体系，其高性价比利于非金融专业向宽客转型，例如程序员通过 CFA 考试转型宽客。同时，该证书在国内有非常高的认可度，有利于更

好地推销产品。CQF（数量金融工程证书）尽管听起来是为宽客量身定做的，但是实际上仅仅适用于大型卖方机构的量化分析岗位，从官网观察到报名该课程的学生多为投资银行、商业银行和评价机构的在职人员，而且该证书在国内名气并不大。

## 1.7 宽客的两大阵形：P 宗与 Q 宗

P 宗与 Q 宗之争，如同金庸小说《笑傲江湖》里面的华山派气宗和剑宗针锋相对一样，充满着话题。

董可人在其文《我是高频交易工程师》中提道，Q 宗根本在于风险中性测度，即不存在无风险套利机会，换句话说可以完美对冲各种风险。因为其定价是基于不存在无风险套利的，这是一个非常虚幻的假设，其模型要求得到一个很好的数学解析解，必定很注重模型，而对实盘上的历史数据不太重视。所以 Q 宗宽客“重模型而轻数据”，涉及的数学知识主要是随机过程、偏微分方程等分支。

P 宗的“P”是指真实概率测度。所谓真实，主要指模型依赖的概率分布是从历史数据上估算出来的。最多只能说是从真实数据上估算出来的，显然没有什么东西保证历史一定会重演（比如黑天鹅）。从定义可以看出这套方法主要依赖数据，数据量越大估算的效果越好，表现为“重数据而轻模型”。在实践中，对冲基金或者各大投行券商的自营部门拿到一组数据时，会用若干备选模型来“跑”，由计算结果来选择最佳的模型，涉及的技术主要是计量、时间序列、更加复杂的统计学习/机器学习。不难看出，为了倒腾数据，这套方法练到上层就要开始“刷装备”。在电子化时代这最终演化为拼机房的“军备”竞赛。

从应用上来讲，Q 宗是模型固定，用数据来精化模型的参数；而 P 宗则可以有若干备选模型，由数据的计算结果来选择最佳的模型。

Q 宗可以让你在缺少数据的情况得出一些结论，从而可以凭空制造一些东西出来，所以卖方（投行）用来做衍生品定价，业务模式是开发新的衍生品出来卖



出去。P 宗则喜欢数据量大，这天然就是买方所需要的技术，因为他们本来就需要针对大量证券做出筛选和投资决策，业务是数据驱动的。

从区别就可以看出两者在发展方向上的不同。本质上说，Q 宗属于“制造业”，大家比的就是造出更多更好的衍生品来卖，但如果生产出来的东西没人买，生意显然就做不下去。而 P 宗其实属于“服务业”，那些数据技术不会给你创造出什么新产品，而是通过对本来就存在的业务（比如投资决策）进行精细加工来达到优化的目的。

## 1.8 宽客的 3 种职能分类

P 宗与 Q 宗之争在国内呈现出“P”强“Q”弱的局势，大部分的量化投资者都集中在买方机构，其中又以私募为主。相对而言，在买方机构，宽客可以更好地根据其职能的不同来分类：量化 IT 工程师、量化研究员和量化交易员。

### 1.8.1 量化 IT 工程师

量化 IT 工程师（Quant Developer）是金融领域的“码农”，隶属于 IT 团队，日常协助研究员、交易员解决编程技术难题，必须精通 C++ 语言，并且运用 C++ 语言搭建低延迟交易系统，对撮合机制、订单类型、交易接口、市场微观结构等有一定的理解。

相对于软件业和互联网行业的“码农”，量化 IT 工程师更加纯粹。一般意义上的“码农”，做的工作重复度高，很多时候是从一种语言换到另一种语言，从一个框架换到另一个框架。而且，抛去技术本身就具有挑战性，一个项目成败的关键点往往与技术能力无关，需要紧贴市场潮流。社交软件火了，扎堆于相关软件的研发，轮到手游火了，又加班加点赶项目。同理，编程语言潮流日新月异，可能刚刚学会了 Java，就有人开始鼓吹 PHP，等“扑”过去后，却发现已经是 iOS 的天下了。

量化 IT 工程师则仅依靠 C++ 语言一路披荆斩棘就可以了,而且行业风向变化周期较长,有足够的时间专研,例如追求更高效地利用 CPU 多进程、更高速的算法、更低的网络延时等。而且,这种集金融学、数学、计算机科学等多种学科于一身的工作,对于沉迷于技术世界的人来说本身就有特别的吸引力。

自动化交易系统通常分三个模块:交易策略模块、事件驱动模块、周边支持模块(比如网络、操作系统、数据库等)。交易越高频,复杂度就往后两个模块倾斜;越低频,复杂度就往第一个模块加深。所以在高频交易机构,量化 IT 工程师的作用会越发显著。

## 1.8.2 量化研究员

量化研究员(Quant Researcher)会注重自动化交易系统的第一个模块,即交易策略模块。日常工作是维护已经运行的策略,并且开发新的策略。策略产生的交易信号容易失效,而且它的生命周期也是有限的,可能上半年是赚钱的,到了下半年却不赚钱,因此对于已在实盘“跑”的策略需要经常调整参数。

而另一项任务就是研发新的策略,首先需要经过样本内外的检验,一些关键指标,如夏普比率、年化收益和最大回撤都要达到要求,然后进行模拟盘测试,只有两者表现出色,才可以在实盘上“跑”:开始是小资金测试,表现出色后慢慢扩大资金容量。

根据交易策略的类型确定资金容量。原则上是交易越高频,资金容量越低。高频交易策略的夏普比率可以达到喜人的十几,但是其策略资金容量只有几亿元。虽然国债套利的年化收益较低,但其可管理的资金量是令人恐怖的。在相同交易频率的策略中,最看重夏普比率。高的夏普比率意味着可以运用杠杆来放大收益。

在追求高夏普比率的同时需要注意避免过拟合。过拟合就是在建模时把历史数据的噪声也拟合进去了,这会造成尽管在历史回测的时候效果很好,但是一上实盘就亏损。所以,有效地优化策略参数是判断研究员是否优秀的重要标准。当前行业共识是线性模型要优于非线性模型,精英因子要优于巨型因子。



基于编程语言易学性和数据分析的需求，量化研究员一般会主攻 Python、R、MATLAB 等动态语言，并且需要一定的 SQL 基础，若想往高频交易方向发展，则要求懂 C++ 语言。

### 1.8.3 量化交易员

国内绝大部分交易员只是接收客户或基金经理的指令，在一定时间内，尽可能低买高卖或者分批操作来降低冲击成本，因为只是执行，又叫执行交易员。根据证券品种不同，可以分为下面几种交易员。

- **股票交易员**：熟悉几百个股票代码，而且手要快，快速执行基金经理下达的指令。
- **债券基金交易员**：由于债券市场并不是竞价交易，能否成交、成交的价格更多地取决于交易员在银行界和投资界的社会资源。
- **期货交易员**：闻得出盘面的气味，一个建仓或平仓大单，十个人能做出十个价。
- **外汇宏观交易员**：具有深厚的经济学功底，能够以宏观、估值、资金流等看得见、摸得着的因素揣摩市场方向。

交易员若得到基金经理的赏识，有管理一部分资金的权力，可在承担一定风险的情况下通过主观判断或者量化策略得到超额收益。量化交易员（Quant trader）就属于这种具有主观决策权的交易员。量化交易员不仅要懂市场，而且需要有把交易思路转换成代码的能力，推荐语言也是 Python 语言。

一些策略的特性决定它只能半自动交易，如期货差价套利和期权波动性套利，需要综合考虑最新的信息，如新政策、基本面信息、历史均值回归数据等，在开盘前设置参数，盘中监视操作界面，当出现突发事件，也需要改成人工操作。

尽管从功能上看，量化交易员和量化研究员比较类似，但存在根本上的区别，量化交易员的本质是交易员，而量化研究员的本质是宽客。当然，他们的努力方向是相同的，都需要在统计学、数学、金融和编程多方面加深研究，通过多年水滴石穿的行业积累，逐渐把握住赚钱的本质。



## 1.9 宽客的四大派系

《第一财经日报》的《宽客在中国》系列报道指出，美国次贷危机导致 2009 年很多华尔街的资深宽客回国发展，掀起了量化投资热潮。经过两年多的发展，量化投资在国内市场上已经形成券商、公募基金、私募基金及期货市场四大派系。虽然这是从行业上划分的，但在策略运用及交易风格上，这四大派系也存在不小的差异。

目前，券商行业中从事量化研究与投资的人占多数，而且根据不同的目标客户形成了一家券商内部共存几个量化投资团队的局面。

2009 年，指数型基金发行井喷，指数基金经理一将难求。同年 13 家基金公司推出了 15 只主动管理型量化基金，总规模约 241.04 亿元人民币，仅占公募基金总规模的 1% 左右。但值得注意的是，这些基金业绩排名都相对靠后。

私募一直可谓“船小好调头”，虽说也存在诸多限制，但在量化投资方面也开始了较早的试水。特别是股指期货出炉后，私募向对冲基金的转型尤为便捷。

期货市场也是量化投资的主要阵营。国内期货市场最早的量化投资者以现货商为主，主要做一些内外盘的对冲套利。2008 年熊市之后，市值保值需求凸显，一些私募开始对一些与期货相关的证券进行套保。此外，期货公司也纷纷成立了金融工程部。

目前量化投资者主要人群集中在期货公司、私募基金及券商的自营、基金公司的专户。但是在规模上看，以私募基金为主要参与群体。

### 1.9.1 券商资管

2011 年 3 月，国泰君安发行了券商集合理财产品中的第一只量化对冲基金——国泰君安君享量化，一日售罄，募集资金 5.08 亿元人民币。随后的 7 月~8 月，更是创下一月内连发 5 只对冲基金的发行纪录。继国泰君安之后，中航证券、东方证券和浙商证券也陆续发行量化集合理财产品，到 2012 年，10 只量化券商理财





产品的规模为 16 亿元人民币左右。虽然，相比起整个券商资产管理行业近 300 只产品和 1300 多亿元的规模，券商宽客们仅仅分食了一小块的蛋糕，但是，券商宽客的力量正在成长。

除公开的集合理财产品外，一些券商的资产管理部也在非公开发行的客户资产管理领域进行量化投资尝试，为客户设计个性化投资产品。

## 1.9.2 公募基金

早在 2004 年光大保德信基金就推出了首只采用量化策略的基金——光大保德信量化核心。但起初阵营扩张非常缓慢，直到 2009 年开始才出现加速迹象。

目前量化投资在公募产品中的应用主要包括三个方面：第一是利用量化投资管理纯被动指数基金，控制跟踪指数误差；第二是主动管理的增强指数量化基金，以富国基金为代表；第三是主动管理的量化产品，择时、选股等都通过量化模型完成。

这三个方面的应用其实也反映了量化投资的三个层次，选股是最低的层面，之后是行业配置，最上面是仓位选择。在纯被动指数基金方面，采用量化投资的成本较低。对于已成立运作的指数基金来说，在系统建立起来后，相同管理类型的产品都可以共用一套系统，基金经理的工作实际上就是对细节进行微调。增强指数基金其实也是一种主动管理基金，只是不进行择时，仓位和一般指数基金相同，始终不高于 95%，不低于 90%。增强主要是通过对股票和行业的选择来实现的。主要通过 Alpha 因子量化模型选择股票和行业。完全主动的量化基金相当于增强指数基金的“升级版”，在利用量化模型选择股票和行业的同时，增加择时模型，股票、行业和仓位都通过量化模型来决定。

不过，公募基金量化产品的数量和规模只是市场上的“冰山一角”。

### 1.9.3 私募基金

与公募基金和券商资产管理部门在量化投资领域的谨小慎微、亦步亦趋不同，私募宽客的生命力犹如百折不挠的野草，只要有一丁点的土壤和空气，种子就会拼命汲取营养冒出头来。

早在 2002 年，原君安证券出身的康晓阳在美国接触了量化投资之后，回国就建立了首个分析师数据库，2006 年 9 月 11 日，康晓阳又成立了深圳天马资产管理有限公司，其发行的产品深国投·天马就是当前有记录的最早一只阳光量化私募产品。

私募形式灵活、投资标的多样，是宽客发展的最好土壤。而且私募做量化的逻辑也相对更简单，它的目标就是获取收益，因此哪怕是揪住了任何一点能赚钱的东西，都会孜孜不倦地研究下去并把其转化为收益。而一旦这个 Alpha 因子消失了，它也会发了疯一样地去寻找另一个。而这种力量，则是公募基金和券商资管等基金管理人所缺乏的，因为他们管理的并不是自己的钱，对他们来说，量化只是工作或者事业；而对于很多量化私募来讲，这是他们谋生的根本。

而且，目前来看，私募在量化投资上是最为积极的，中国未来宽客中的“大奖章基金”很可能就在民间私募中出现。

### 1.9.4 期货市场

国内期货市场最早的量化投资者主要是一批经常做内外盘对冲的现货商，如伦敦铜和国内铜之间的内外盘套利交易。

随着 A 股市场在 2008 年进入熊市，市值保值需求凸显，一些私募基金开始对一些与期货相关的证券进行套保。但当时市场环境还比较“原始”，主要在一些行情软件上搭载一些技术分析指标，对风险控制、资金分配管理等方面的研究和规范化也比较薄弱。随着股指期货的推出，期货市场的关注度逐渐提高。

因为当时只有商品期货，所以期货市场的关注度较低，总体上算是小众市场。



此外由于许多品种定价权受制于国外，隔夜的跳空较为常见，所以量化投资存在较大的风险。在股指期货推出来以后，期货市场受到了社会的广泛关注。而期货市场的整体成交量也出现了大幅度的增长，多个品种成交金额居于世界前列，这为量化投资奠定了良好的基础。

与此同时，期货行业也有了不小变化。期货公司纷纷成立了金融工程部或者投资咨询部，证券研究所与期货研究所联系更为紧密，更注重金融产品的研究；客户对于程序化交易开始逐渐认同；市场上各种量化投资平台蜂拥而出。



# 第 2 章

## Python 量化编程基础

本章首先介绍了 Python 语言作为量化交易入门语言的理由，然后讲解了 Python 的基础概念，介绍常用的数据分析库 NumPy 与 Pandas，以及机器学习库 scikit-learn，最后讲解绘图库 Matplotlib 的基本用法。

纯粹从计算机性能和速度出发，C++语言无疑是量化交易的最佳选择，它是最接近计算机底层的语言。C++语言通过对 CPU 多核多线程的优化，合理分配内存及显卡计算等，可充分“榨干”硬件的每一份性能，在微秒必争的高频交易世界去追求更低的延时。在高频交易领域，只有更快的速度才能抢到价格更低的单子，交易成本就会低于对手。所以只要交易数量足够大，积少成多，其赢利是“恐怖”的。

为了保证交易系统的高效性和稳健性，公司会习惯让 IT 团队自主开发或者购买 C++搭建的底层通用平台开发程序，对内部组件进行封装，对外提供行情和交易 API 接口，可以让其他编程语言，如 Python、C#和 R 等语言接入来开发策略。

但从新手入门量化交易的角度来看，Python 语言则打败强大的 C++语言，成为新手最受欢迎的语言，原因有以下几点。

(1) 在高频交易领域，必须靠速度来赚钱。但是对于中低频的交易，速度就



显得不那么重要了，一定程度的网络延时或者交易系统延时还是能接受的，赢利重心转向交易策略。所以 C++ 语言的特性在中低频交易中并不能凸显出来。

(2) 在研发交易策略上，C++ 语言不是一个很好的选择。作为静态语言，什么都要自己定义，这会浪费大量的精力。打个比方，拿到一份股票数据，不管是分析历史价格趋势还是波动性，第一件事就是通过画图来直观感受。Python 语言有非常强大的数据分析库，不到十行代码就能够轻松搞定。若用 C++ 语言的话，肯定会花大量的时间在编译、调试、再编译上。当最后把图画出来，发现已经码完上百行代码了。所以用 C++ 语言去研发策略的话，大部分的精力容易先浪费在“造轮子”上，之后才会考虑策略本身的问题。人的精力和天赋是有限的，很难同时兼顾数学建模和底层代码调试这种差距巨大的工作。而 Python 语言则提供很多造好的“轮子”，例如，科学计算库 NumPy 与 Pandas、数据绘图库 Matplotlib、技术指标计算库 TA-Lib 等。这些“轮子”是用强大的 C++ 语言编写并且封装好的，在需要的时候直接调用就可以了。Python 胶水语言的特性可以让研究员放心地把精力放在数学建模上。

(3) C++ 操作的烦琐也间接反映其学习曲线过于陡峭，对于量化交易新手而言非常不友好。与之相反，Python 因为其语法非常像英语，学习的边际成本会大幅度降低。Python 容易到连小学生都可以学，例如为了响应 2017 年国务院发布《新一代人工智能发展规划》，山东省最新出版的小学信息技术六年级教材加入了 Python 语言的学习内容。

综上所述，Python 倚着其易学性、强大的数据分析库，以及可扩展性成为新手入门量化交易的首选语言。

## 2.1 Python 运行环境搭建

因大多数初学者是 Windows 用户，所以本次 Python 运行环境的搭建是基于 Windows 的，并且在以后讲述的章节中用到 vn.py 1.9.0 版本，Python 的安装版本为 Python 2.7（用户亦可直接安装 vn.py 1.9.2 LTS 版本，二者相差不大）。

## 2.1.1 安装 Anaconda2-5.0.0 (32 位)

Anaconda 是一个基于 Python 的环境管理工具，其中包含了 Conda，Python，NumPy，Scipy，Jupyter Notebook 在内的超过 180 个科学库及其依赖项。

Conda 是包及其依赖项和环境的管理工具，适用于 Python，R，Ruby，Lua，Scala，Java，JavaScript，C/C++ 和 Fortran 语言。Conda 可以用于快速安装、运行和升级包及其依赖项，在计算机中便捷地创建、保存、加载和切换环境。因为 Conda 同样是一个环境管理器，仅需要几条命令，就可以创建一个完全独立的环境来运行不同的 Python 版本，同时支持在常规的环境中使用常用的 Python 版本。

由于兼容 `vn.py` 的需要（详见第 3 章），要求安装特定版本，即 Anaconda2 5.0.0-Windows-x86，如图 2-1 所示。



图 2-1 Anaconda 安装界面

这里要说明一下，Anaconda2 5.2.0 对应的是 `vn.py` 1.9.0，也是本书所使用的版本。若使用 `vn.py` 1.8.1 或者以前的版本，则需安装 Anaconda2-4.0.0，另外在 2019 年推出 `vn.py` 2.0 版本后，将会兼容 Anaconda3 64 位的版本。

安装完成后要进入 Anaconda Prompt，在 Prompt 上可以使用 `conda` 或者 `pip` 命令来安装对应的 Python 依赖库，如图 2-2 所示。



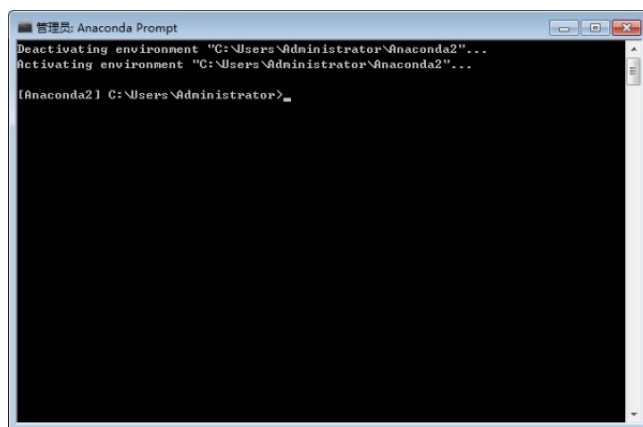


图 2-2 Anaconda Prompt 界面

## 2.1.2 设置 Anancoda 环境

安装 nb\_conda 用于 Jupyter Notebook 自动关联 nb\_conda 的环境，如图 2-3 所示。

```
conda install nb_conda
```

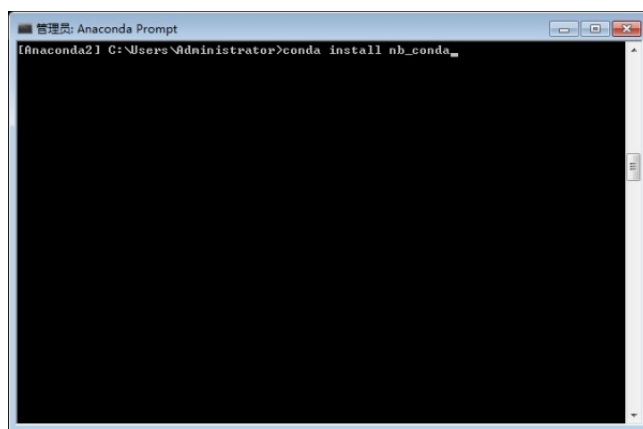


图 2-3 安装 nb\_conda

在终端中使用：

```
conda create -n env_name package_names
```

在上面的命令中, `env_name` 是所创建环境的名称, `package_names` 是用户要安装在环境中的包名称。

当同时使用 Python 2.x 和 Python 3.x 中的代码时这很有用。创建具有特定 Python 版本的环境, 例如, 创建环境名称为 `py2`, 并安装 Python2, 在终端中输入:

```
conda create -n py2 python=2
```

创建环境名称为 `py3`, 并安装最新版本的 Python3, 在终端中输入:

```
conda create -n py3 python=3
```

由于用户做的项目不同, 有时候会用到 Python2, 有时候会用到 Python3。所以可以在自己的计算机上创建了这两个环境, 并分别取名: `py2`, `py3`。这样用户可以根据不同的项目轻松使用不同版本的 Python。

最后, 如果用户要安装特定版本 (例如 Python 3.6), 输入下面命令, 如图 2-4 所示。

```
conda create -n py3.6 python=3.6
```

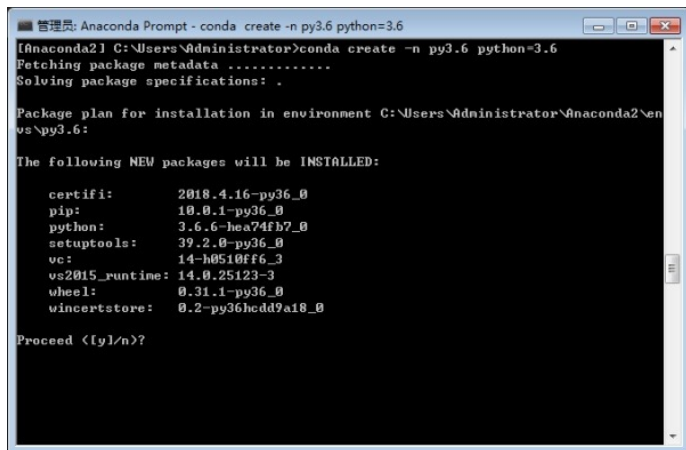


图 2-4 创建 Python 环境命令



### 2.1.3 创建共享环境

共享环境的作用在于能够让其他用户在安装的创作者代码中使用所有包，并确保这些包的版本正确。比如，A 开发了一个药店数据分析系统，提交给项目部署系统的 B 来部署，但是 B 并不知道 A 当时开发时使用的是 Python 哪个版本，以及使用了哪些包和包的版本。为解决这个问题，A 可以在当前的环境终端中使用：

```
conda env export > environment.yaml
```

将 A 当前的环境保存为 yaml 文件（包括 Python 版本和所有包的名称），让 B 及其他人更轻松地安装 A 的代码的所有依赖项，如图 2-5 所示。

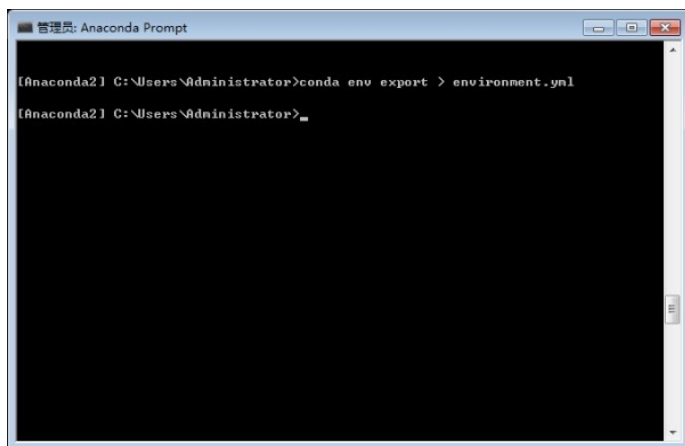


图 2-5 创建共享环境

### 2.1.4 列出共享环境

如图 2-6 所示，输入下面命令可以列出共享环境：

```
conda env list
```

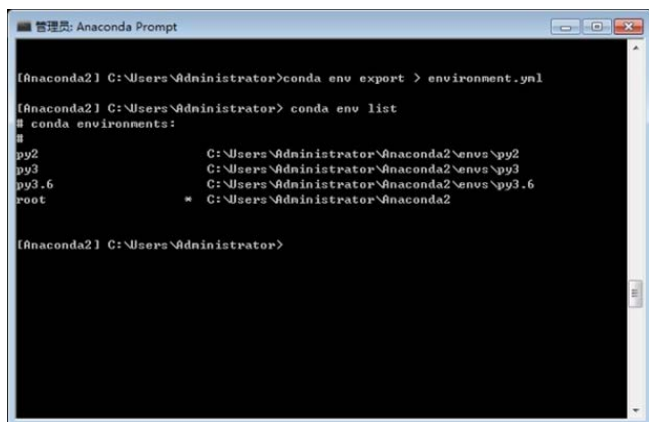


图 2-6 列出共享环境

## 2.1.5 安装 Jupyter Notebook

Jupyter Notebook 是基于网页的用于交互计算的应用程序，它可以在网页页面中直接编写和运行代码，代码的运行结果也会直接在代码块下显示。在编程过程中如果需要编写说明文档，可在同一个页面中以 Markdown 的格式编写。安装 Jupyter Notebook 的命令：

```
conda install jupyter notebook
```

安装完后，在 Anaconda Prompt 中输入命令“jupyter notebook”，按 Enter 键便以网页的形式打开。单击“new”选项就会显示创建出来的 Python 环境，如图 2-7 所示。



图 2-7 Jupyter Notebook 界面上的 Python 环境



图 2-7 所示的 Python[conda root]和 Python[default]指的是 Anaconda 默认环境，即 Python2。

在图 2-7 右上角“new”选项单击“Python[conda env:py3]”进入新的页面，可输入第一行 Python 代码“Hello world”，单击图 2-8 所示方框的运行按钮或用快捷键“Ctrl+Enter”即可执行代码。

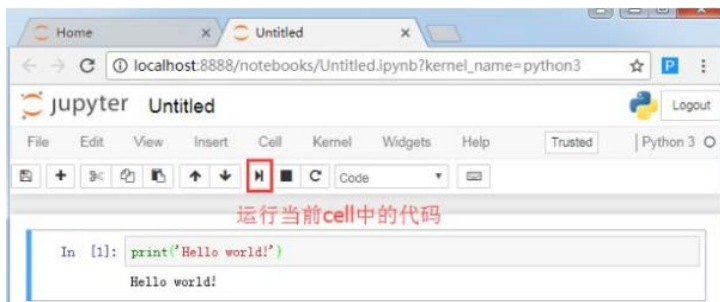


图 2-8 运行按钮

安装好 Python 后就可以开始正式学习了。为了更好掌握 Python 语言的基本用法，这里将 Python 语言分成 4 部分逐一讲解，分别是数据、函数、条件判断和循环。

## 2.2 数据

数据分为字符串、数字、容器、布尔值和空值 5 种类型。在代码里，数据是用变量存放的，而数据又有 5 种类型，所以在给变量取名时，建议用“变量名称+数据类型”的形式，而且要记住数字不能放在变量名称的开头，Python 是区分大小写的。

### 2.2.1 字符串

字符串（String）是由数字、字母、下画线组成的一串字符，用单引号（'）

或者双引号 ( " ) 括起来。字符串是不可改变类型的。

需要注意的是, Python 2 默认的编码格式是 ASCII 格式, 只有在文件开头加入 “# encoding: UTF-8”, 修改编码格式才能正确地打印汉字, 否则在读取中文时会报错。而 Python 3 源码文件默认使用 UTF-8 编码, 所以可以正常解析中文。

```
# 合约类型常量
PRODUCT_EQUITY = u'股票'
PRODUCT_FUTURES = u'期货'
PRODUCT_OPTION = u'期权'
PRODUCT_INDEX = u'指数'
PRODUCT_COMBINATION = u'组合'
PRODUCT_FOREX = u'外汇'
PRODUCT_UNKNOWN = u'未知'
PRODUCT_SPOT = u'现货'
```

Python 支持使用反斜杠 ( \ ) 转义特殊字符, 如表 2-1 所示。

表 2-1 转义特殊字符

转义字符	描 述	转义字符	描 述
( 在行尾时 )	续行符	\n	换行
\	反斜杠符号	\v	纵向制表符
'	单引号	\t	横向制表符
"	双引号	\r	回车
\a	响铃	\f	换页
\b	退格 ( Backspace )	\oyy	八进制数, yy 代表的字符, 例如: \o12 代表换行
\e	转义	\xyy	十六进制数, yy 代表的字符, 例如: \x0a 代表换行
\000	空	\other	其他字符以普通格式输出

在实际操作中, 例如导入 Excel 表时, Excel 表里的数据其实是字符串( String ), 必须转换成数字数据类型才能进行下一步的数据分析。



## 2.2.2 数字

数字（Number）数据类型用于存储数值。指定一个值时，Number 对象就会被创建。数字数据类型一旦改变就会被分配一个新的对象。

数字数据类型分为两种。

### 1. 整数（int）

在 32 位系统上，整数的位数为 32 位，取值范围为  $-2^{31} \sim 2^{31}-1$ ，即 -2147483648 ~ 2147483647。在 64 位系统上，整数的位数为 64 位，取值范围为  $-2^{63} \sim 2^{63}-1$ ，即 -9223372036854775808 ~ 9223372036854775807。

### 2. 浮点数（float）

浮点数用来处理实数，即带有小数的数字。类似于 C 语言中的 double 类型，占 8 个字节（64 位），其中 52 位表示底，11 位表示指数，剩下的一位表示符号。

```
#数据类型
EMPTY_INT = 0           #浮点型
EMPTY_FLOAT = 0.0       #整数型

#交易所推送过来的 TICK 行情 K 线数据中，高开低收为浮点型，成交量为整数型
open = EMPTY_FLOAT      #开盘价
high = EMPTY_FLOAT      #最高价
low = EMPTY_FLOAT       #最低价
close = EMPTY_FLOAT     #收盘价

volume = EMPTY_INT      # 成交量
openInterest = EMPTY_INT # 持仓量
```

## 2.2.3 容器

容器把不同的数据类型放在一起方便使用，根据用途不同可分为 4 种，分别是列表、元组、集合和字典。

## 1. 列表

列表 (List) 可以完成大多数集合类的数据结构所实现的工作。列表中元素的类型可以不同, 支持数字、字符串甚至可以包含列表 (所谓嵌套)。列表是写在方括号 “[ ]” 之间、用逗号分隔开的元素列表。和字符串一样, 列表同样可以被索引和截取, 列表被截取后返回一个包含所需元素的新列表。

```
# 参数列表, 保存了参数的名称
paramList = ['name',
             'className',
             'author',
             'vtSymbol']

# 变量列表, 保存了变量的名称
varList = ['inited',
           'trading',
           'pos']

# 同步列表, 保存了需要保存到数据库的变量名称
syncList = ['pos']
```

列表的基本操作包括查询列表长度, 增加、删除和修改元素, 以及进行正序和倒序查询等。举例说明如下。

(1) 定义 5 个元素的列表。

```
foodList = ['肠粉', '糯米鸡', '马蹄糕', '双皮奶', '鹤鹑蛋烧卖']
```

正序	0	1	2	3	4
倒序	-5	-4	-3	-2	-1

(2) 通过正序和倒序来查询列表元素。

```
foodList = ['肠粉', '糯米鸡', '马蹄糕', '双皮奶', '鹤鹑蛋烧卖']

foodList[4]    # 查询第 5 个元素
>> '鹤鹑蛋烧卖'

foodList[-1]   # 查询倒数第 1 个元素
>> '鹤鹑蛋烧卖'
```



(3) 查询列表长度，增加、删除和修改元素。

```
# 1.查询列表长度
len = len(foodList)
len
>> 5

# 2.列表操作: 增加元素
foodList.append('艇仔粥')
foodList
>> ['肠粉', '糯米鸡', '马蹄糕', '双皮奶', '鹤鹑蛋烧卖', '艇仔粥']

# 3.列表操作: 删除元素
del foodList[2]    #删除第3个元素
foodList
>> ['肠粉', '糯米鸡', '双皮奶', '鹤鹑蛋烧卖', '艇仔粥']

# 4.列表操作: 修改元素
foodList[0]='豆浆' #修改第1个元素为'豆浆'
foodList
>> ['豆浆', '糯米鸡', '双皮奶', '鹤鹑蛋烧卖', '艇仔粥']
```

(4) 切片访问，分别访问前3个元素和后3个元素。

```
# 1.访问前3个元素
foodList[0:3]
>> ['豆浆', '糯米鸡', '双皮奶']

# 2.简化版访问前3个元素
foodList[:3]
>> ['豆浆', '糯米鸡', '双皮奶']

# 3.访问后3个元素
foodList[-3:]
>> ['双皮奶', '鹤鹑蛋烧卖', '艇仔粥']
```

## 2. 元组

元组 (Tuple) 是一系列不可变的 Python 对象。元组是一种序列，就像列表一样。元组和列表之间的主要区别是元组不能像列表那样改变元素的值，可以简



单地理解为“只读列表”。需要注意的是，元组使用小括号“( )”，而列表使用方括号“[ ]”，对比如下：

```
# a.列表
foodList = ['肠粉', '糯米鸡', '马蹄糕', '双皮奶', '鹤鹑蛋烧卖']

# b.元组
foodTup = ('肠粉', '糯米鸡', '马蹄糕', '双皮奶', '鹤鹑蛋烧卖')
```

### 3. 集合

集合是一个无序不重复元素的序列。之前说的列表可以包括重复的元素，而集合是没有重复元素的容器，用花括号“{ }”来表示集合，其操作如下：

```
#定义集合，交易所
ExchangeSets={'中金所','上期所','郑商所','大商所','上交所','深交所'}
print (ExchangeSets)
>> {'上期所', '深交所', '郑商所', '上交所', '中金所', '大商所'}
```

集合的基本功能是进行成员关系的测试，以及删除重复元素，可以使用大括号“{ }”或者 set()函数来创建，注意：定义一个空集合必须用 set()，而不是“{ }”，因为“{ }”是用来创建一个空字典的。集合的基本操作举例说明如下。

```
# 1.定义一个空的集合
BitcoinExchangeSets=set()

# 2.使用 update 来增加一个元素
BitcoinExchangeSets.update(['OKCOIN 比特币交易所','火币比特币交易所','LBANK 比特币交易所'])
print (BitcoinExchangeSets)
>> {'OKCOIN 比特币交易所', 'LBANK 比特币交易所', '火币比特币交易所'}

# 3.使用 discard 来删除一个元素
BitcoinExchangeSets.discard('OKCOIN 比特币交易所')
print (BitcoinExchangeSets)
>> {'LBANK 比特币交易所', '火币比特币交易所'}

# 4.使用 in 查询关键字
txBool='火币比特币交易所' in BitcoinExchangeSets
print (txBool)
```



```
>> True

# 5 修改集合内的元素
#第一步, 删除
BitcoinExchangeSets.discard('火币比特币交易所')
#第二步, 增加
BitcoinExchangeSets.update(['OKEX 比特币交易所'])
print (BitcoinExchangeSets)
>> {'LBANK 比特币交易所', 'OKEX 比特币交易所'}
```

#### 4. 字典

字典是另一种可变容器模型，可存储任意类型的对象。字典的每个键值对“key=>value”用冒号“:”分隔，每个对之间用逗号“,”分隔，整个字典包括在花括号“{ }”中，格式为 d = {key1 : value1, key2 : value2 }。键必须是唯一的，且是不可变的，但值则不必唯一，可取任何数据类型，如字符串、数字或元组。字典的基本操作举例说明如下。

##### (1) 创建新的字典。

```
# 创建新的字典
ExchangeDict = {'中金所': 'CFFEX',
                '上期所': 'SHFE',
                '郑商所': 'CZCE',
                '大商所': 'DCE',
                '国际能源交易中心': 'INE'}
```

##### (2) 增加、删除、查询和修改元素。

```
# 1. 增加元素
ExchangeDict['上期所']=['SGE']
ExchangeDict
>> {'上期所': 'SHFE',
     '上期所': ['SGE'],
     '中金所': 'CFFEX',
     '国际能源交易中心': 'INE',
     '大商所': 'DCE',
     '郑商所': 'CZCE'}
```



```
# 2. 删除元素
del ExchangeDict['上金所']
ExchangeDict
>> {'上期所': 'SHFE', '中金所': 'CFFEX', '国际能源交易中心': 'INE', '大商所': 'DCE',
'郑商所': 'CZCE'}

# 3. 查询元素, 根据交易所名称查询交易所代码
ExchangeDict['中金所']
>> 'CFFEX'

# 4. 修改元素
print ('修改前, 上期所代码: ', ExchangeDict['上期所'])
ExchangeDict['上期所'] = ['SQS']
print ('修改后, 上期所代码: ', ExchangeDict['上期所'])
>> 修改前, 上期所代码:  SHFE
修改后, 上期所代码:  ['SQS']
```

## 2.2.4 布尔值

在 Python 中布尔值 (Bool) 通过常量 “True” 和 “False” 来表示。比较运算符如 “<” “>” “==” 等返回的类型就是布尔类型; 布尔类型通常在 if 和 while 语句中应用。应该注意的是, 布尔是 int 的子类 (继承 int), 故判断如 True == 1 False == 0 时, 其返回结果是 True。

布尔值的应用举例说明如下。导入金融库 TA-Lib 的简单均线函数 SMA, 定义新的函数 sma, 若 array = True, 则输出一系列均线, 反之, 若 array = False, 则输出最新的一条均线数据。

```
import talib
def sma(self, n, array=False):
    """简单均线"""
    result = talib.SMA(self.close, n)
    if array=True:
        return result
    return result[-1]
```

一般情况下, “if array:” 已经是默认 “if array = True:”, 所以函数可以进一步简化, 如下:



```
def sma(self, n, array=False):
    """简单均线"""
    result = talib.SMA(self.close, n)
    if array:
        return result
    return result[-1]
```

## 2.2.5 空值

空值 (None) 是 Python 语言里一个特殊的值，表示的是一个空对象，但不能将其理解为 0，因为 0 是有意义的。None 既可以被赋值给任何变量，也可以将任何变量赋值给 None。所以 None 常用于初始化。

```
self.datetime = None # python 的 datetime 时间对象
```

## 2.3 函数

函数是实现某个特定的功能，可以重复使用的代码块。Python 提供了许多内置函数，比如 print()。用户既可以根据自己的需求自行创建函数，即用户自定义函数，又可以调用第三方库的函数/模块。下面重点介绍自定义函数和第三方库的函数/模块。

### 2.3.1 自定义函数

下面代码演示了先定义函数，再调用定义好的函数实现其功能的过程。

```
'''
定义函数
函数功能: 两个数相加
输入: x,y 是两个要输入的数字
输出: z 是两个数相加的和, 用 return 键来导出
'''
# 1. 定义函数
def add (x,y):
```

```
    z = x + y
return z

# 2.调用函数
a=1
b=2
c=add(a,b)
print ('a 和b相加为',c)
a 和b相加为 3
```

### 2.3.2 第三方库的函数

这里运用 2.2.4 节用过的例子来说明如何调用第三方库的函数。首先用 `import talib` 调用金融库 TA-Lib，再结合 TA-Lib 自带的函数 `talib.SMA()` 来定义新的函数 `SMA`。

```
import talib
def sma(self, n, array=False):
    """简单均线"""
    result = talib.SMA(self.close, n)
    if array=True:
        return result
    return result[-1]
```

## 2.4 条件判断

Python 条件语句是通过一条或多条语句的执行结果（例如：True 或者 False）来决定代码块的执行。

`if` 语句的判断条件可以用 `>`（大于）、`<`（小于）、`==`（等于）、`>=`（大于或等于）、`<=`（小于或等于）来表示。由于 Python 并不支持 `switch` 语句，所以多个条件判断，只能用 `elif` 来实现。用 `or`（或）时表示两个条件有一个成立时判断条件成功。用 `and`（与）时，表示只有两个条件同时成立的情况下，判断条件才成功。当 `if` 有多个条件时可使用括号来区分判断的先后顺序，括号中的判断优先



执行，此外 `and` 和 `or` 的优先级低于 `>`（大于）、`<`（小于）等判断符号，即大于和小于在没有括号的情况下要优先判断。

下面是条件判断操作：

```
# 1.简单条件判断
# 《摔跤吧，爸爸》，豆瓣评分
scoreNum = 9.1
if scoreNum >= 8 :
    print ('我要看这部电影')
else:
    print ('电影太烂，不想看')
>> 我要看这部电影

# 2. 多个条件判断
age = int (input ('请输入你家狗狗的年龄，按 Enter 键获取计算结果: '))
if age < 0:
    print ('狗狗的年龄不能少于 0 岁')
elif age == 1:
    print ('相当于 14 岁的人')
elif age == 2:
    print ('相当于 14 岁的人')
else:
    human = 22+ (age - 2)*5
    print ('对应人类年龄: ', human )
>> 请输入你家狗狗的年龄，按 Enter 键获取计算结果: 5
对应人类年龄: 37
```

下面的代码是更加复杂的条件判断。判断的格式为“非 `a` 为真且 `b` 为真”，在这里，`a=self.inited = False`，非 `a = True`，所以逻辑判断简化为：当 `count >= 100` 为真时，初始化状态变成 `True`。

```
self.inited = False          #初始化状态默认为 False
self.count += 1
if not self.inited and self.count >= 100:
    self.inited = True       #当 count >= 100 时，初始化状态变成 True
```

## 2.5 循环

循环分为 while 循环和 for 循环。While 循环在给定的判断条件为 True 时执行循环体，否则退出循环体；而 for 循环仅仅重复执行语句。因此 while 循环可以与 for 循环结合使用，此时称作嵌套循环。

在循环中，一般采用 if 或 if...else 或 if...elif...else 语句作为判断条件，也可以使用专门的循环控制语句来更改语句执行的顺序。Python 支持的循环控制语句有 3 种，分别是：

- (1) break 语句，在语句块执行过程中终止当前循环，并且跳出整个循环；
- (2) continue 语句，在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环；

- (3) pass 语句，空语句，是为了保持程序结构的完整性。

for 循环和 while 循环的区别在于：

- (1) for 循环是在序列穷尽时停止的，while 循环是在条件不成立时停止的。
- (2) for 一般不会出现死循环，而 while 容易写成死循环。
- (3) for 循环语句申明循环变量，while 循环语句需判断循环条件。

循环的典型操作如下所示。

- (1) 创建列表，for 循环遍历元素。

```
# 1.容器：一天中吃几次饭
eatList = ['吃第一次饭','吃第二次饭','吃第三次饭']
# 循环
for i in eatList:
    print (i)
>> 吃第一次饭
吃第二次饭
吃第三次饭
```





(2) 遍历字典里的元素，把股票代码全部改成大写。

```
# 2.对字典进行循环
'''
定义字典: 6家公司(GAFATA)的股票
key是公司名称, value是公司代码
'''
gafataDict = {'腾讯':'HK:00700','阿里巴巴':'baba','苹果':'Apple','谷歌':
':GOOGLE','Facebook':'fb','亚马逊':'amzn'}
# 将股票代码全部改成大写(upper)
# 注意用 key, value
for key,value in gafataDict.items():
    newValue = value.upper()
    gafataDict[key]= newValue
print (gafataDict)
>> {'腾讯': 'HK:00700', '阿里巴巴': 'BABA', '苹果': 'APPLE', '谷歌': 'GOOGLE',
'Facebook': 'FB', '亚马逊': 'AMZN'}
```

(3) 用 continue 跳出当前循环。

```
# 3.continue 用于跳出当前循环
'''
定义字典: 6家公司(GAFATA)的股票
key是公司名称, value是公司代码
'''
gafataDict = {'腾讯':'HK:00700','阿里巴巴':'baba','苹果':'Apple','谷歌':
':GOOGLE','Facebook':'fb','亚马逊':'amzn'}

# 注意用 key, value
for key,value in gafataDict.items():
    if key == '苹果':
        continue
print ('当前公司',key,'当前股票代码: ',value)
>> 当前公司 腾讯 当前股票代码: HK:00700
当前公司 阿里巴巴 当前股票代码: baba
当前公司 谷歌 当前股票代码: GOOGLE
当前公司 Facebook 当前股票代码: fb
当前公司 亚马逊 当前股票代码: amzn
```

(4) 用 break 退出整个循环。

```
# 4.break 用于退出整个循环
gafataDict = {'腾讯':'HK:00700','阿里巴巴':'baba','苹果':'Apple','谷歌':
'GOOGLE','Facebook':'fb','亚马逊':'amzn'}

# 注意用 key, value
for key,value in gafataDict.items():
    if key == '苹果':
        print ('当前公司: ',key,'当前股票代码: ',value)
        break
print ('现在公司',key,'其股票代码: ',value)
>> 现在公司 腾讯 其股票代码: HK:00700
现在公司 阿里巴巴 其股票代码: baba
当前公司 苹果 当前股票代码: Apple
```

## 2.6 类和实例

面向对象编程中最重要的概念就是类（Class）和实例（Instance）。简单地说，类是抽象的概念，而实例是根据类创建出来的一个个具体的“对象”，每个对象都拥有相同的方法，但各自的数据可能不同。打个比喻，类是“猫”，而实例可以分别是“波斯猫”“英国短毛猫”“加拿大无毛猫”等。每个品种都有作为猫的共同属性，但是它们又有各自不同的特征。

类名通常采用驼峰式命名方式，即混合使用大小写字母来命名，尽量让字面意思体现出类的作用。Python 采用多继承机制，一个类可以同时继承多个父类。

继承的基类有先后顺序，写在类名后的圆括号“()”里。它也默认继承 object 类，因 object 类是所有类的基类。

下面用代码来分别实现类和实例这两个概念，以便理解。

### 2.6.1 定义学生父类

定义学生父类代码如下所示。



```

In[1]
# encoding: UTF-8
class Student(object):
    '''定义学生父类'''

    #类的属性
    studCount = 0                #初始化学生人数，以便后面统计
    classroom = '102'            #教室号
    kindergarten = '双叶幼儿园' #地点
    name = None                  #学生名字
    age = 0                      #学生年龄

    #实例方法-----
    def __init__(self, name, age):
        '''类的初始化'''
        self.name = name        #学生名字
        self.age = age          #学生年龄
        self.studCount += 1     #用于统计学生人数
        self.__headmaster = '高仓文太' #私有属性

    def head(self):
        '''园长'''
        print 'The Kindergarten headmaster : ',self.__headmaster

    def displayCount(self):
        '''学生人数统计'''
        print "Total Student %d" % self.studCount

    def displayStudent(self):
        '''学生信息'''
        print "Name : ", self.name, ", Age: ", self.age

    #类的方法-----
    @classmethod
    def displayClassroom(cls):
        '''教室号'''
        print "Classroom is %s" % cls.classroom

    #静态方法-----
    @staticmethod
    def displayKindergarten( ):

```

```
'''学校名'''  
print "Kindergarten is %s" % Student.kindergarten
```

类的属性就是在类中定义的变量，这个属性虽然归类所有，但类的所有实例都可以访问到。在类外，可以通过类和实例访问公有的属性；而对于私有属性，即任何以双下划线开头的名字，只能在该类中被调用，不能在外部调用或者继承，不能被子类继承，也不会被子类覆盖，可以使用“实例名.类名私有属性名”进行访问，如 `a._Student__headmaster`。

实例方法只能被实例对象调用，类的方法与普通的函数有一个特别的区别——它们必须有一个额外的名称放在前面，按照惯例这个名称是 `self`。

`init` 是初始化方法，用于设置实例的相关属性。在创建类的实例的时候一般会自动调用这个方法，对实例的属性进行初始化。`init()` 方法的第一个参数永远都是 `self`，表示创建实例本身，在 `init` 方法内部，可以把各种属性绑定到 `self`，因为 `self` 指向创建实例本身。

类的方法由类调用，采用 `@classmethod` 装饰，至少传入一个 `cls`（代指类本身，类似 `self`）参数。执行类方法时，自动将调用该方法的类赋值给 `cls`。

静态方法由类调用，无默认参数。将实例方法参数中的 `self` 去掉，在方法定义上方加上 `@staticmethod`，就成为静态方法。它属于类，与实例无关。

类的方法的第一个参数是类对象 `cls`，通过 `cls` 引用的必定是类对象的属性和方法；而实例方法第一个参数是实例对象 `self`，通过 `self` 引用的可能是类属性，也有可能是实例属性，不过在相同名称的类属性和实例属性均有的情况下，实例属性的优先级更高。

## 2.6.2 定义父类实例

定义父类实例代码如下。

```
In[2]  
a=Student('野原新之助',5) #定义实例
```



```

a.classroom          #访问公共属性,用点号 . 来访问对象的属性
a._Student__headmaster  #访问私有属性,用点号 . 来访问对象的属性
a.head()             #园长名字
a.displayCount()     #学生人数
a.displayStudent()   #学生信息
a.displayClassroom() #教室号
a.displayKindergarten() #学校名
Out[2]
'''分别输出下面信息'''
101                                #访问公共属性
高仓文太                          #访问私有属性
The Kindergarten headmaster : 高仓文太 #园长名字
Total Student 1                   #学生人数
Name : 野原新之助 , Age: 5        #学生信息
Classroom is 102                  #教室号
Kindergarten is 双叶幼儿园       #学校名

```

继承类的构造方法有以下两种。

- (1) 经典类的写法：父类名称.init(self,参数 1,参数 2,...)。
- (2) 新式类的写法：super(子类,self).init(参数 1,参数 2,...)。

在定义子类的构造函数时，只有先继承再构造，才能继承父类的属性。

子类的方法如果和父类的方法重名，子类就会覆盖掉父类，这就是继承“多态”。调用方只管调用，不管细节，而新增一个子类时，只要确保新方法编写正确，而不用管原来的代码，这就是著名的“开闭”原则。

- (1) 对扩展开放：允许新增子类；
- (2) 对修改封闭：不需要修改依赖父类的函数。

### 2.6.3 定义团体子类

定义团体子类代码如下。

```

In[3]
class Group(Student):

```

```
'''定义团体子类'''

#新增类的属性
Class = '向日葵班级'
Teacher = '吉永老师'

#继承类的构造
def __init__(self,name, age):
    '''类的初始化'''
    super(Group, self).__init__(name, age)

#新增实例方法
def favorite(self):
    '''类的初始化'''
    if self.name == '野原新之助':
        print u'小新最喜欢动感超人了'

#重构旧的实例方法
def displayCount(self):
    '''学生人数统计'''
    print "人数是 %d" % self.studCount
```

## 2.6.4 定义子类实例

定义子类实例代码如下。

```
In[4]
b=Group('野原新之助',5)      #定义子类实例
b.classroom                 #访问父类的属性
b.Teacher                   #访问子类新增的属性
b.favorite()                #子类新增方法
b.displayCount()            #子类的方法如果和父类的方法重名，子类会覆盖掉父类

Out[4]
'''分别输出下面信息'''
'102'                       #父类的属性
'吉永老师'                  #子类的属性
小新最喜欢动感超人了        #子类新增方法
人数是 1                    #子类重构方法
```



通过上面的例子，可以总结出面向对象编程的三个基本特征，如下所述。

（1）封装：将抽象得到的数据和行为（或功能）相结合，形成一个有机的整体（即类）；封装的目的是增强安全性和简化编程，使用者不必了解具体的实现细节，而只要通过外部接口、特定的访问权限来使用类的成员。

（2）继承：在遵循原有设计和既有代码不变的前提下，添加新功能，或改进算法。记住其“开闭”原则是对扩展开放，对修改封闭。

（3）多态：所有继承子类应该能直接引用父类，这样可以把复杂类型的公用部分剥离出来，形成稳固的抽象类。其他引发变化的相似因素则被分离成多个子类，以确保单一职责原则得到遵守，并能相互替换。

## 2.7 NumPy 与 Pandas

下面介绍 2 个常用的数据分析库：NumPy 与 Pandas。

NumPy 可以理解成 Python 的矩阵处理器，含有强大的  $N$  维数组对象 Array，具有比较成熟的函数库，用于整合 C/C++ 和 Fortran 代码的工具包、实用的线性代数、傅立叶变换和随机数生成函数。NumPy 提供很多方便计算统计的功能，可以计算平均值 `mean()` 与标准差 `std()`。NumPy 支持向量计算，例如向量相加，与标量相乘。

Pandas 是基于 NumPy 的数学分析工具，是为了解决数据分析任务而创建的。由 AQR 资本管理公司于 2008 年 4 月开发，并于 2009 年年底开源出来。Pandas 最初被作为金融数据分析工具而开发出来，因此，Pandas 为时间序列分析提供了很好的支持。Pandas 的名称来自面板数据（Panel data）和数据分析（Data analysis）两个名称的缩写。这一工具可用来存储和处理大型矩阵，比 Python 自身的嵌套列表结构要高效得多。



## 2.7.1 一维数组

NumPy 中一维数组的格式是 Array, Pandas 中的则是 Series。所以, 在开始操作一维数组前, 需要先搞清 List (列表), Array 和 Series 的区别。

(1) Array 与 List 的区别: NumPy 数组 Array 中的每一个元素都必须是同一种数据类型, 而列表 List 中的元素可以是不同的数据类型。

(2) Array 与 Series 的区别: Series 有索引, 可以用 index 来定义这个索引, 方便索引后面的元素。Array 则没有。

下面开始操作一维数组。

### 1. NumPy 一维数组

先通过 `import NumPy as np` 导入 NumPy 库, 创建一维数组, 可以像列表一样查询单个元素, 也可以切片访问。`dtype` 则用于显示其一维数组的数据类型。

```
# 导入 NumPy 包, 并且命名为 np
import NumPy as np
# 导入 Pandas 包, 命名为 pandas
import Pandas as pd

# 1. 定义一组数据 array([ ])
a = np.array([2,3,4,5,6])

# 2. 查询
a[0]
>> 2

# 3. 切片
a[1:3]
>> array([3, 4])

# 4. 数据类型 dtype
a.dtype
dtype ('int32')
```

相对于列表, NumPy 提供统计分析功能, 如用 `std()` 计算标准差, 用 `mean()`



计算平均值，同时也支持向量运算。代码如下：

```
# 1.平均值
a.mean()
>> 4.0

# 2.标准差
a.std()
>> 1.4142135623730951

b = np.array([1,2,3])
c = b*4
c
>> array([ 4,  8, 12])
```

## 2. Pandas 一维数组

Pandas 一维数组在定义的时候就有索引 index，格式是 `x=pd.Series([ ],index[ ])`。定义好一维数组以后，就可以使用 `describe()` 方法来获取一维数组的描述统计信息，包括一维数组里面有多少个元素、平均值是多少、标准差及四分位。

有 2 种方法来查询一维数组，分别是 `iloc` 和 `loc`，但是其用法有一点区别：

(1) `iloc` 可以用 “index=[ ]” 根据元素的位置来获取元素的值。

(2) `loc` 根据索引的值来获取元素的值。

具体操作如下：

```
import pandas as pd
# 1.定义股票数组
stocks = pd.Series([54.74,190.9,173.14,1050.3,181.86,1139.49,],
                    index= ['腾讯',
                             '阿里巴巴',
                             '苹果',
                             '谷歌',
                             'Facebook',
                             '亚马逊'])
```

```
# 2. 查看一维数组 Series 的统计特性
stocks.describe()
>> count      6.000000
mean      465.071667
std      491.183757
min      54.740000
25%     175.320000
50%     186.380000
75%     835.450000
max     1139.490000
dtype: float64
# 3. iloc 查询
stocks.iloc[0]
>> 54.74

# 4. loc 查询
stocks.loc['Facebook']
>> 181.86
```

跟 NumPy 一维数组 Array 一样, Pandas 的 Series 也支持向量运算, 但只能与索引值相同的值相加。

代码如下:

```
# 1. 向量运算, 向量相加
s1 = pd.Series([1,2,3,4], index= ['a','b','c','d'])
s2 = pd.Series([10,20,30,40], index= ['a','b','e','f'])
s3 = s1 + s2
s3
>> a    11.0
b    22.0
c     NaN
d     NaN
e     NaN
f     NaN
dtype: float64
```

在数据分析过程中如果不希望出现空值 NaN, 有 2 个办法:

(1) 用 `dropna()` 方法来删除空值 NaN。



(2) 可用 `add` 将两个一维数组相加, 并传入 `fill_value` 参数, 其中 `fill_value` 用来填充空值, 比如用 0 来填充。

```
# 2.删除空值 NaN
s3.dropna()
>> a    11.0
b    22.0
dtype: float64

# 3.用数字 0 来填充空值
s3 = s1.add(s2,fill_value=0)
s3
>> a    11.0
b    22.0
c     3.0
d     4.0
e    30.0
f    40.0
dtype: float64
```

## 2.7.2 二维数组

相对于一维数组, 二维数组更加常见, 因为二维数组既有行也有列, 类似于 Excel 里面的二维表格。在 NumPy 中是通过 `Array` 来创建一个二维数组的, 其数组中的所有元素都是同样的数据类型 (如都是数字, 不能包含如商品名称、生产日期等其他数据类型), 虽然容易计算, 但不利于表达像 Excel 这样的情况。Pandas 则是通过 `DataFrame` 来创建一个二维数组的, 有利于表达像 Excel 中的数据。下面分别介绍。

### 1. NumPy 二维数组

(1) 构建 NumPy 二维数组。

```
# 导入 NumPy 和 Pandas
import NumPy as np
import Pandas as pd

# 在 NumPy 中创建二维数组,
```

```
# 主要在([ ])中包含一系列 [ ]  
a = np.array([  
    [1,2,3,4],  
    [5,6,7,8],  
    [9,10,11,12]  
])
```

(2) 查询二维数组中的元素。这里提供 3 种方法，代码如下：

```
# 方法一：查询数组中的某个元素  
a[0,2]  
>> 3  
  
# 方法二：查询数组中的某行，其中用“:”来代替任意列  
a[0,:]  
>> array([1, 2, 3, 4])  
  
# 方法三：查询数组中的某列，其中用“:”来代替任意行  
a[:,0]  
>> array([1, 5, 9])
```

(3) 分组计算，使用数轴参数“axis”，其中 axis=1 是按每一行进行分组计算的；axis=0 是按每一列进行分组计算的，代码如下：

```
# Numpy 数组的数轴参数 'axis'  
a.mean(axis=1)  
>> array([ 2.5,  6.5, 10.5])  
  
a.mean(axis = 0)  
>> array([5.,  6.,  7.,  8.])
```

## 2. Pandas 二维数组

相对于 NumPy 二维数组的 Array，Pandas 二维数组 DataFrame 有两个优点：

(1) 数组中的每一列都可以是不同类型，方便表示 Excel 中的数据。

(2) 数组中的每一行/列都有一个索引表格，类似于一维数组 Series，使得常见的表格数据很容易制作。



具体操作如下。

(1) 构建 DataFrame, 其中, 字典后跟{ },大括号中有中括号[], 下面数据中没有“'”的是数据, 可以计算平均值, 代码如下。

```
import pandas as pd
# 首先定义一个字典, 映射列名于相应列的值。

510050Dict = {
    '日期': ['2018-08-01', '2018-08-02', '2018-08-03'],
    '开盘价': ['001616528', '001616528', '0012602828'],
    '最高价': [236701, 236701, 236701],
    '收盘价': ['强力 VC 银翘片', '清热解毒口服液', '感康'],
    '最低价': [6, 1, 2],
    '成交量': [82.8, 28, 16.8],
}
# 然后导入有序字典, 将 salesDict 定义成有序字典
from collections import OrderedDict
salesOrderedDict = OrderedDict(salesDict)
# 最后传入数据框: 传入字典, 列名
salesDf = pd.DataFrame(salesOrderedDict)

salesDf
```

>>	购药时间	社保卡号	商品编码	商品名称	销售数量	应收金额	实收金额
0	2018-01-01	周五	001616528	236701	强力 VC 银翘片	6	82.869.00
1	2018-01-02	周六	001616528	236701	清热解毒口服液	1	28.024.64
2	2018-01-06	周三	0012602828	236701	感康	2	16.815.00

(2) DataFrame 可分别运用 iloc 和 loc 查询数组中的元素。具体查询代码如下。

```
# 1. 平均值, 按照每一列来计算平均值
salesDf.mean()
>> 商品编码    236701.000000
销售数量      3.000000
应收金额      42.533333
实收金额      36.213333
dtype: float64

# 2. 数据框中的 iloc 根据位置来获取数据的值, 第一位是行, 第二位是列
# 查询某一元素的值, 用[]表示行号列号
salesDf.iloc[0,3]
```

```
>> '强力 VC 银翘片'

# 3. 分别查询某一行和某一列
salesDf.iloc[0,:]
购药时间    2018-01-01 周五
社保卡号      001616528
商品编码      236701
商品名称      强力 VC 银翘片
销售数量        6
应收金额      82.8
实收金额      69
Name: 0, dtype: object

salesDf.iloc[:,0]
>> 0    2018-01-01 周五
1    2018-01-02 周六
2    2018-01-06 周三
Name: 购药时间, dtype: object

# 3. 数据框中的 loc 是根据行和列的索引值（行名称或者列名称）来获取数据的
# 其行名称默认为行号
salesDf.loc[0, '商品名称']
>> '强力 VC 银翘片'

salesDf.loc[0,:]
>> 购药时间    2018-01-01 周五
社保卡号      001616528
商品编码      236701
商品名称      强力 VC 银翘片
销售数量        6
应收金额      82.8
实收金额      69
Name: 0, dtype: object

salesDf.loc[:, '商品名称']
>> 0    强力 VC 银翘片
1    清热解毒口服液
2    感康
Name: 商品名称, dtype: object
```



```
# 4. 查询特定的列
salesDf[['商品名称', '应收金额']]
>> 商品名称  应收金额
0    强力 VC 银翘片  82.8
1    清热解毒口服液  28.0
2    感康  16.8

# 5. 切片功能: 指定范围
# 通过冒号指定范围来获取元素
salesDf.loc[:, '商品名称': '实收金额']
>>
商品名称  销售数量  应收金额  实收金额
0    强力 VC 银翘片    6    82.8 69.00
1    清热解毒口服液    1    28.0 24.64
2    感康    2    16.8 15.00
```

(3) 通过条件判断来筛选数据, 先构建一个查询条件, 然后根据查询条件来筛选出符合查询条件的行, 代码如下。

```
# 1. 构建查询条件
querySer = salesDf.loc[:, '销售数量'] > 1

# 2. 查看数组类型
type(querySer)
>> pandas.core.series.Series

# 3. 查看判断结果
querySer
>> 0    True
1    False
2    True
Name: 销售数量, dtype: bool

# 4. 应用查询条件来筛选出结果为 True 的行
salesDf.loc[querySer, :]
>>
   购药时间  社保卡号  商品编码  商品名称  销售数量  应收金额  实收金额
0  2018-01-01 周五  001616528   236701  强力 VC 银翘片    6    82.8 69.0
2  2018-01-06 周三  0012602828  236701    感康        2    16.8 15.0
```



## 2.8 scikit-learn 机器学习库

机器学习算法可以通过在提供的数据中进行自学习而变得更聪明，得出更符合人们需求的结果。例如，从市场上的橘子随机抽取一些样本（训练数据），清洗样本，标记其物理属性（特征），同时要记录着这些橘子是甜的，还是不甜的（标签）。

机器学习算法会随着时间把结果越变越好，因为更多的训练数据意味着模型会被训练得更准确，并且会自行修正错误的预测。最重要的是可以用同样的机器学习算法训练出不同的模型。

scikit-learn 是一个用于 Python 的免费开源机器学习库。它提供了现成的功能来实现诸如线性回归分类器、SVM、k-均值和神经网络等多种算法，还有一些可直接用于训练和测试的样本数据集。由于其速度、鲁棒性和易用性，它是许多机器学习应用程序中使用最广泛的库之一。

其安装过程也非常简单，下面介绍 2 种安装方式。

### （1）pip 安装

```
pip install -U scikit-learn
```

### （2）conda 安装

```
conda install scikit-learn
```

### 2.8.1 机器学习的步骤

#### 1. 提出问题

机器学习的目标是解决我们生活工作中遇到的问题，如喜欢看《龙珠》的用户还会喜欢看什么样的动漫。



## 2. 理解数据

数据理解主要包括 3 方面内容：

- (1) 采集数据，根据研究问题采集相关数据。
- (2) 导入数据，来源包括如 Excel 数据库、网络等。导入数据到 Python 的数据结构中，如 Pandas 的 DataFrame。
- (3) 查看数据的信息，如描述统计信息，并从整体上理解数据。

## 3. 数据清洗

数据清洗就是对数据进行预处理，从数据集中提取出我们想要的东西，即特征。

## 4. 构建模型

通过训练数据来构建模型，将第 3 步提取的特征，放入机器学习算法中构建模型。这是机器学习的核心步骤。

## 5. 评估模型

评估模型就是通过测试数据来评估模型的准确性。

下面通过两个简单的例子，即线性回归和逻辑回归来描述使用机器学习库 scikit-learn 的整体运用过程。

### 2.8.2 线性回归

为了解决相关性的问题，如学习时间是否和考试成绩正相关，机器学习通常会先搞清楚特征和标签的关系：学习时间是特征（类似数学方程式中的自变量  $x$ ），考试成绩是标签（类似数学中的因变量  $y$ ），写成数学表达式即  $y=a+bx$ ，然后分析学习与考试分数的相关性关系。

## 1. 构建字典

(1) 先构建字典, 然后通过有序字典 `OrderedDict()` 对该字典进行排序, 最后把有序字典的数据导入 Pandas 二维数组 `DataFrame` 中, 代码如下。

```
#导入包, 2个
import Pandas as pd
from collections import OrderedDict
# 先构建一个数据集(字典),
# 字典格式是 Dict = {'xx':[ ], 'yy':[ ]},其中x和y是一一对应关系
examDict={
    '学习时间':[0.50,0.75,1.00,1.25,1.50,1.75,1.75,2.00,2.25,
                2.50,2.75,3.00,3.25,3.50,4.00,4.25,4.50,4.75,5.00,5.50],
    '分数':    [10, 22, 13, 43, 20, 22, 33, 50, 62,
                48, 55, 75, 62, 73, 81, 76, 64, 82, 90, 93]
}
# 然后变成有序字典, 用OrderedDict()
examOrderDict = OrderedDict(examDict)

# 通过有序字典构建数据框pd.DataFrame
examDf = pd.DataFrame(examOrderDict)
examDf.head()
>> 学习时间 分数
0    0.50    10
1    0.75    22
2    1.00    13
3    1.25    43
4    1.50    20
```

(2) 提取特征(X)和标签(y)。

```
# 提取特征(x)和标签(y)
exam_X = examDf.loc[:, '学习时间']
exam_y = examDf.loc[:, '分数']
```

(3) 绘制散点图, 很直观地查看其相关性情况, 即学习与成绩具有强正相关性, 散点图如图 2-9 所示。

```
# 先导入Matplotlib
import matplotlib.pyplot as plt
```



```
# 散点图 scatter, 其中习惯用大写 X 表示特征, 小写 y 表示标签
plt.scatter(exam_X, exam_y, color='b', label='exam data')
# 然后在 label 中展开, 增加散点图标签
plt.xlabel('Hours')
plt.ylabel('Score')

# 显示散点图
plt.show()
```

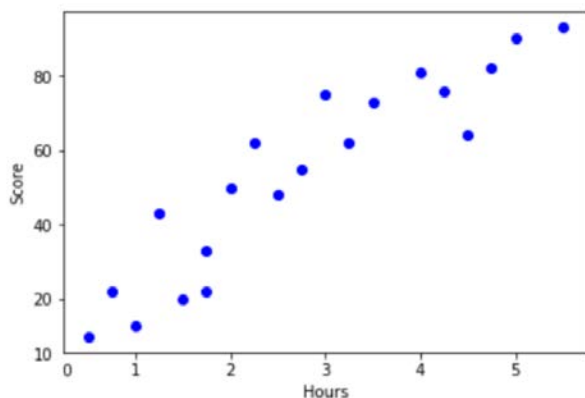


图 2-9 通过散点图直观展示特征和标签的关系

## 2. 建立训练数据和测试数据

交叉验证 (Cross-validation) 指的是在给定的建模样本中, 拿出大部分样本建模型, 留小部分样本用刚建立的模型进行预报。而 `train_test_split` 是交叉验证中常用的函数, 功能是从样本中按比例随机选取训练数据 (train) 和测试数据 (test)。

需要输入以下三个参数。

- 第 1 个参数: 所要划分的样本特征 (X 值)。
- 第 2 个参数: 所要划分的样本标签 (y 值)。
- 第 3 个参数: `train_size` 即训练数据占比, 可以是整数或百分比。如果是整数的话就是样本的数量。图 2-10 所示的是实例中选择了 80% 的训练数据。



图 2-10 训练数据占比

输出以下四个参数。

- 训练数据的特征 `X_train`。
- 测试数据的特征 `X_test`。
- 训练数据的标签 `y_train`。
- 测试数据的标签 `y_test`。

故建立训练数据和测试数据，代码如下：

```
#建立训练数据和测试数据,要按照下面的顺序
X_train, X_test, y_train, y_test = train_test_split(exam_X,
                                                    exam_y,
                                                    train_size=0.8)

#输出数据大小
print('原始数据特征', exam_X.shape ,
      ', 训练数据特征', X_train.shape ,
      ', 测试数据特征', X_test.shape )

print('原始数据标签', exam_y.shape ,
      ', 训练数据标签', y_train.shape ,
      ', 测试数据标签', y_test.shape )
>> 原始数据特征 (20,) , 训练数据特征 (16,) , 测试数据特征 (4,)
原始数据标签 (20,) , 训练数据标签 (16,) , 测试数据标签 (4,)
```

最后绘出训练和测试的散点图，如图 2-11 所示。

```
#绘出训练和测试的散点图
plt.scatter(X_train, y_train, color='b', label='train data')
plt.scatter(X_test, y_test, color='red', label='test data')
#添加图标标签
plt.legend(loc=2)
plt.xlabel('Hours')
```

```
plt.ylabel('Score')
plt.show()
```

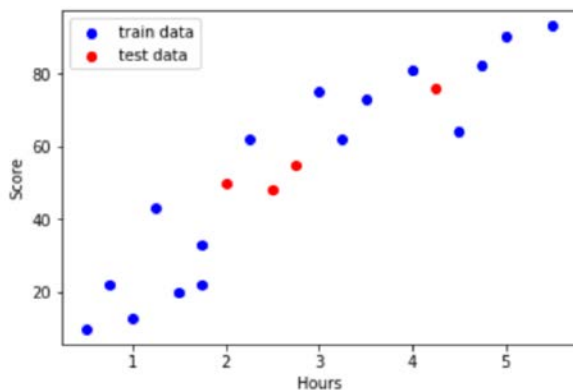


图 2-11 绘制散点图

### 3. 训练模型

(1) 将训练数据特征和标签转换成二维数组  $XX$  行 $\times$ 1 列，用的是 `values.reshape(-1,1)`。

```
'''
scikit-learn 要求输入的特征必须是二维数组的类型，但是我们目前只有 1 个特征，
所以需要转换成二维数组的类型。
错误提示信息: Reshape your data either using array.reshape(-1,1)
if your data has a single feature
'''
#将训练数据特征和标签转换成二维数组 xx 行 $\times$ 1 列，用 values.reshape(-1,1)
X_train=X_train.values.reshape(-1,1)
y_train=y_train.values.reshape(-1,1)
```

(2) 开始训练模型，一共分为 3 步。

```
#第 1 步: 导入线性回归
from sklearn.linear_model import LinearRegression

# 第 2 步: 创建模型，线性回归
model = LinearRegression()
```

```
# 第3步: 训练模型
# fit()函数, 传入第一个参数是训练数据的特征 x, 第二个参数是训练参数的标签 y
model.fit(X_train, y_train)
>> LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

(3) 建立最佳拟合线。最佳拟合线为  $z = +x$ , 其中, 截距为  $a$ ; 回归系数为  $b$ 。

```
# 建立最佳拟合线
a = model.intercept_
b = model.coef_
print ('最佳拟合线, 截距 a=', a, ', 回归系数 b=', b')
>> 最佳拟合线, 截距 a= [7.38860355], 回归系数 b= [[16.42445973]]
```

#### 4. 绘制最佳拟合线

通过绘图库 Matplotlib 来绘制最佳拟合线, 如图 2-12 所示。

```
import matplotlib.pyplot as plt
#训练数据散点图
plt.scatter(X_train, y_train, color='b', label='train data')

# 训练数据的预测 (通过机器学习 sklearn 中的 Linear Regression, 用 model.predict)
y_train_pred = model.predict(X_train)

#绘制最佳拟合线图, 线而不是散点, 不用 scatter, 而用 plot
plt.plot(X_train, y_train_pred, color='black', linewidth='3', label='best line')

# 添加图标签
plt.legend(loc=2)
plt.xlabel('Hours')
plt.ylabel('Score')

#显示图函数 show()
plt.show()
```



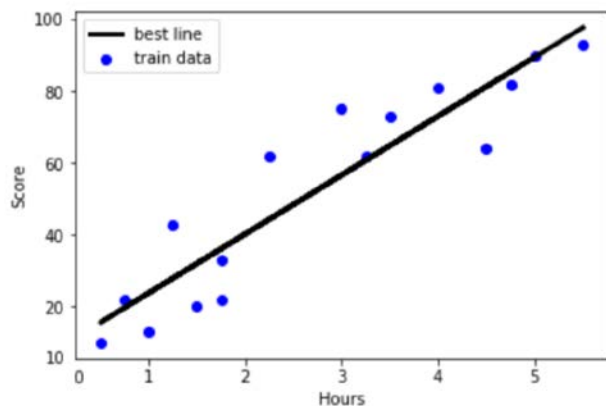


图 2-12 绘制最佳拟合线

## 5. 评估模型

决定系数  $R$  平方为 79%, 代表着 79% 的考试成绩  $y$  的波动可以由回归线描述。

# 1. 相关系数: `corr` 返回结果是一个数据框, 存放的是相关系数矩阵

```
rDf = examDf.corr()
print('相关系数矩阵: ')
```

```
rDf
```

```
>> 相关系数矩阵:
```

```
Out[107]:
```

```
学习时间 分数
```

```
学习时间 1.000000 0.923985
```

```
分数 0.923985 1.000000
```

# 2. 转换矩阵

```
X_test = X_test.values.reshape(-1,1)
```

```
y_test = y_test.values.reshape(-1,1)
```

# 3 线性回归的 `score` 方法得到的是决定系数  $R$  平方

# 评估模型: 决定系数  $R$  平方

```
model.score(X_test,y_test)
```

```
>> 0.7920076607404332
```



## 2.8.3 逻辑回归

逻辑回归（Logistic Regression）模型其实仅在线性回归的基础上，套用了—个逻辑函数，但也正由于这个逻辑函数，使得逻辑回归模型成为机器学习领域—颗耀眼的明星，更是计算广告学的核心。

图 2-13 很直观地展示了逻辑回归的特点。

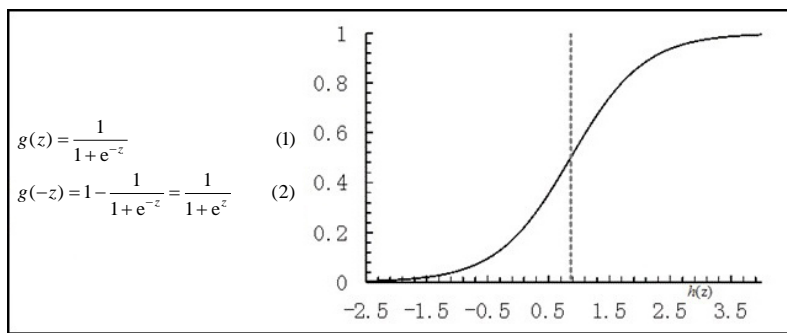


图 2-13 逻辑回归图与其数学公式

其中  $z=a+bx$  即上面学过的线性回归，输出结果为 0 和 1，方便预测其决策结果。下面的例子同样是分析学习与考试分数的相关性，但是此时考试分数只有 0 和 1，分别代表不及格与及格。

### 1. 构建字典

（1）先构建字典，然后通过 `OrderedDict()` 对该字典进行排序，最后把有序字典的数据导入 Pandas 二维数组 `DataFrame` 中。

```
from collections import OrderedDict
import Pandas as pd

# 字典—有序字典—数据框
examDict = {
    '学习时间': [0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 1.75, 2.00, 2.25, 2.50,
                 2.75, 3.00, 3.25, 3.50, 4.00, 4.25, 4.50, 4.75, 5.00, 5.50],
    '通过考试': [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1]}
}
```

```
examOrderdDict=OrderedDict(examDict)
examDf=pd.DataFrame(examOrderdDict)
```

```
#显示前 5 行
```

```
examDf.head()
```

```
>> 学习时间    通过考试
```

```
0    0.50    0
```

```
1    0.75    0
```

```
2    1.00    0
```

```
3    1.25    0
```

```
4    1.50    0
```

(2) 提取特征和标签, 绘制散点图, 如图 2-14 所示。

```
# 1.提取特征和标签
```

```
exam_X = examDf.loc[:, '学习时间']
```

```
exam_y = examDf.loc[:, '通过考试']
```

```
# 2.绘制散点图
```

```
import matplotlib.pyplot as plt
```

```
plt.scatter(exam_X, exam_y, color='b', label='exam data')
```

```
plt.xlabel('Hours')
```

```
plt.ylabel('Pass')
```

```
plt.show()
```

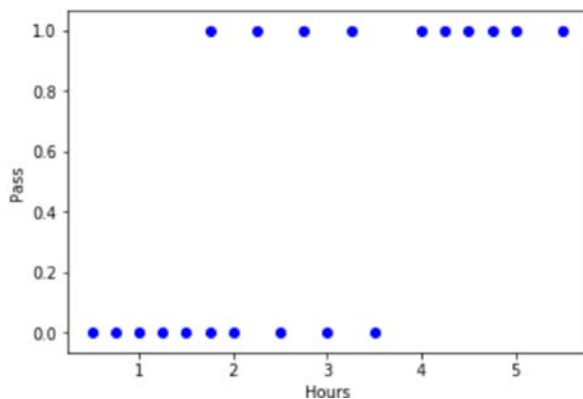


图 2-14 散点图

## 2. 建立训练数据和测试数据

用同样的方法建立训练数据和测试数据。

```
from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test=train_test_split(exam_X,
                                                    exam_y,
                                                    train_size =0.8)

print('原始数据特征: ', exam_X.shape,
      ', 训练数据特征: ', X_train.shape,
      ', 测试数据特征: ', X_test.shape)

print('原始数据标签: ', exam_y.shape,
      ', 训练数据标签: ', y_train.shape,
      ', 测试数据标签: ', y_test.shape)

>> 原始数据特征 (20,) , 训练数据特征 (16,) , 测试数据特征 (4,)
原始数据标签 (20,) , 训练数据标签 (16,) , 测试数据标签 (4,)
```

然后用 Matplotlib 绘制散点图，其中 train data 占 80%，如图 2-15 所示。

```
import matplotlib.pyplot as plt
plt.scatter(X_train, y_train, color='b', label='train data')
plt.scatter(X_test, y_test, color='red', label='test data')

plt.legend(loc=2)
plt.xlabel('Hours')
plt.ylabel('Pass')
plt.show()
```

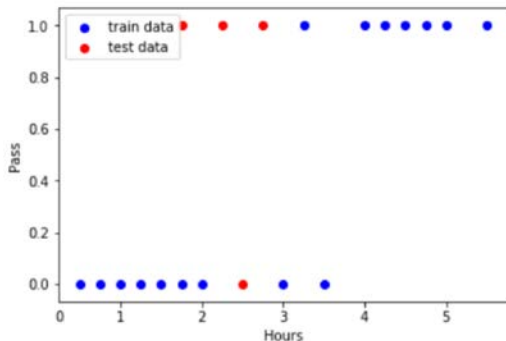


图 2-15 散点图



### 3. 训练模型

训练模型并且建立最佳拟合线。

```
# 逻辑回归函数 LogisticRegression
from sklearn.linear_model import LogisticRegression
X_train = X_train.reshape(-1,1)
X_test = X_test.reshape(-1,1)
Model = LogisticRegression()
model.fit(X_train, y_train)
>> LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0, warm_start=False)
```

### 4. 评估模型

决定系数  $R$  平方为 50%, 代表着 50% 的考试成绩  $y$  的波动可以由回归线描述。

```
model.score(X_test, y_test)
>> 0.5
```

## 2.9 Matplotlib 绘图库

Matplotlib 是 Python 最出名的绘图库, 提供了一整套和 MATLAB 相似的 API, 十分适合交互式绘图。它包含了大量的可用来创建各种图形的工具, 包括简单的散点图、正弦曲线, 甚至是三维图形。Python 科学计算社区经常使用它完成数据可视化的工作。

同时, 为了方便快速绘图, Matplotlib 通过 pyplot 模块提供了一套和 MATLAB 类似的绘图 API, 将众多绘图对象所构成的复杂结构隐藏在这套 API 内部, 只需要调用 pyplot 模块所提供的函数就可以快速绘图并设置图表的各种细节。Matplotlib 库导入 pyplot 模块的命令如下。

```
import matplotlib.pyplot as plt
```

## 2.9.1 用列表绘制线条

利用列表定义  $x$  轴和  $y$  轴，再用 `plt.plot()` 来绘制线条，如图 2-16 所示。

```
import matplotlib.pyplot as plt

#第1步：利用列表信息来定义  $x$  和  $y$  轴坐标
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]

#第2步：使用 plot 绘制线条
#第1个参数是  $x$  的坐标值，第2个参数是  $y$  的坐标值
plt.plot(x, y)

#第3步：显示图形
plt.show()
```

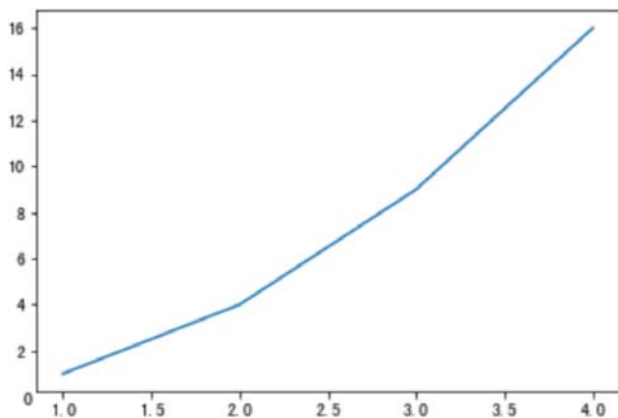


图 2-16 绘制线条

同时，`plt.plot()` 可以设置三种线条属性，属性状态分别如下。

- `color`: 线条颜色，值 `r` 表示红色（red）。
- `marker`: 点的形状，值 `o` 表示点为圆圈标记（circle marker）。
- `linestyle`: 线条的形状，值 `dashed` 表示用虚线连接各点。

```
'''
axis: 坐标轴范围
语法为 axis[xmin, xmax, ymin, ymax],
也就是 axis[x 轴最小值, x 轴最大值, y 轴最小值, y 轴最大值]
'''
plt.axis([0, 6, 0, 20])
plt.show()
```

设置线条属性如图 2-17 所示。

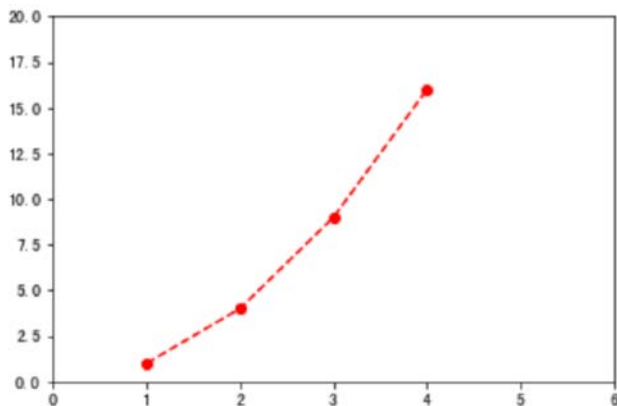


图 2-17 设置线条属性

如果 Matplotlib 输入的参数只能是列表的话，这对数据处理很不利。一般，用户的习惯是传入 NumPy 数组（实际上，所有入参的值内部都会转换为 NumPy 数组）。

## 2.9.2 用数组绘图

下面介绍如何使用 NumPy 数组来绘图，具体步骤如下所述。

(1) 等差数列: NumPy 的函数 `arrange()` 可以生成一个等差数列 `arrange([start, ] stop, [step, ])`，分别对应的是最小值、最大值和等差数。

```
import Numpy as np
t = np.arange(0, 5, 0.2)
t
```

```
>> array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4,
          2.6, 2.8, 3. , 3.2, 3.4, 3.6, 3.8, 4. , 4.2, 4.4, 4.6, 4.8])
```

(2)同时绘制3条线:先分别定义3条线,如图2-18所示。注意运算符,“\*\*”返回 $x$ 的 $y$ 次幂,例如 $10^{**}20$ 表示10的20次方。然后在`plot()`上传入定义好的 $x$ 轴、 $y$ 轴即可。

```
# 线条1
x1 = y1 = t

# 线条2
x2 = x1
y2 = t**2

# 线条3
x3 = x1
y3 = t**3

# 3条线绘图
LineList = plt.plot(x1, y1,
                    x2, y2,
                    x3, y3)

#用 setp 方法可以同时设置多条线条的属性
plt.setp(LineList, color='blue')
plt.show()
```

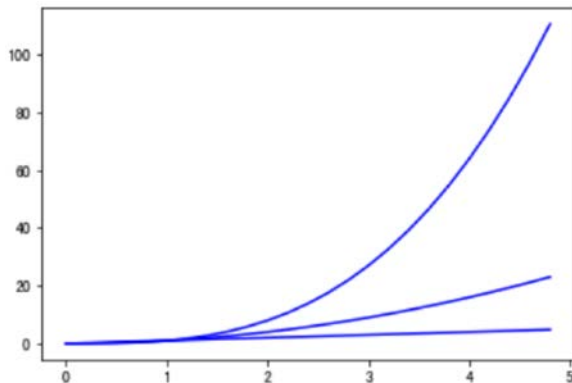


图 2-18 绘制3条线

(3) 在图上添加文本：同样先完成绘图，如图 2-19 所示。然后设置  $x$  轴、 $y$  轴名称和标题名，通过 `annotate()` 添加注释，其中参数名 “ $xy$ ” 表示注释中箭头所在位置，参数名 “ $xytext$ ” 表示注释文本所在位置，`arrowprops` 用于在 “ $xy$ ” 和 “ $xytext$ ” 之间绘制箭头，`shrink` 表示注释点与注释文本之间的图标距离，如图 2-20 所示。

```
# 第 1 步：定义  $x$  和  $y$  坐标轴上的点定义
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]

# 第 2 步：使用 plot 绘制线条
# 第 1 个参数是  $x$  的坐标值，第 2 个参数是  $y$  的坐标值
plt.plot(x,y)
```

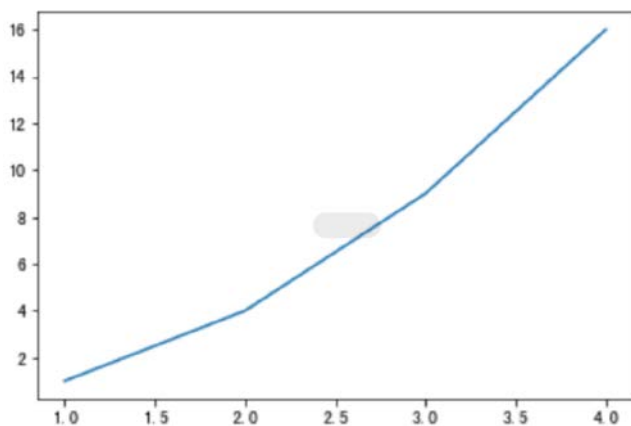


图 2-19 绘制线条

```
# 第 3 步：添加注释
plt.xlabel('x 坐标轴')
plt.ylabel('y 坐标轴')
plt.title('标题')

plt.annotate('这里是注释', xy=(2, 5), xytext=(2, 10),
            arrowprops=dict(facecolor='black', shrink=0.01),)
plt.show()
```



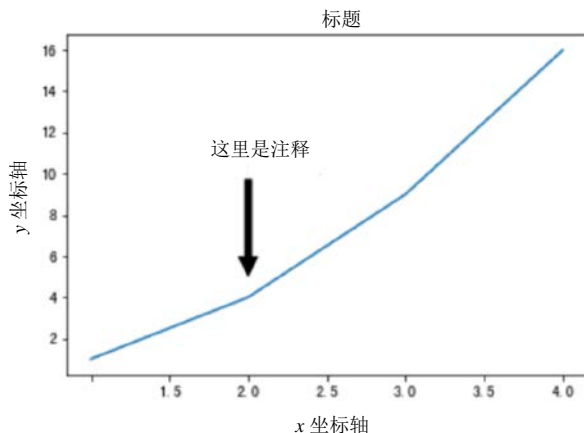


图 2-20 添加注释

### 2.9.3 多个图的绘制

(1) 创建画板:

```
fig=plt.figure(1)
```

(2) 在画板里建立若干个画纸(子图)。在画纸中, subplot()方法的括号里包括三个数字:前两个数字代表要生成几行几列的子图矩阵,第三个数字代表选中的子图位置。例如: subplot(2,1,1)生成一个 2 行 1 列的子图矩阵,如图 2-21 所示。

```
#创建画纸, 并选择画纸 1
'''
等价于 ax1=fig.add_subplot(211)
'''
ax1 = plt.subplot(2, 1, 1)

#在画纸 1 上绘图
plt.plot([1, 2, 3])

#选择画纸 2
ax2 = plt.subplot(2, 1, 2)
#在画纸 2 上绘图
```

```
plt.plot([4, 5, 6])  
  
plt.show()
```

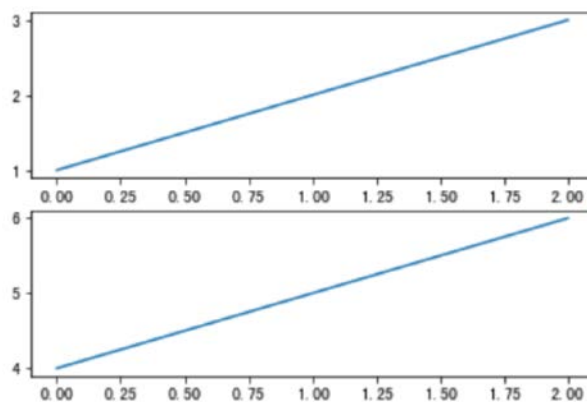


图 2-21 2 行 1 列的子图

# 第 3 章

## vn.py 入门

本章主要讲述 vn.py 交易系统的概况、安装步骤、主交易界面的功能，以及 vn.py 应用框架的结构，包括底层接口、中层引擎和上层应用，最后再对这三层结构的原理做一个具体说明。

### 3.1 vn.py 介绍

vn.py 是基于 Python 语言的量化交易系统，2015 年诞生之初只是单纯对基于 C++ 程序化交易 API 接口进行 Python 封装。随着业内关注度的上升和社区不断的贡献，目前已经一步步成长为一套全面的交易程序开发框架，用户群体也日渐多样化，包括私募基金、证券自营和资管、期货资管及子公司、高校研究机构、个人投资者等。

vn.py 设计之初的目标受众是具有成熟的交易理念、能够稳定赢利的交易员，而这些交易员迫于编程水平有限，无法运用底层语言（如 C++/Java 语言）直接接入原生 API 写出策略。vn.py 的出现大幅度降低了交易员的编程门槛，Python 的易用性，使得交易员即使缺乏 IT 团队的支持，也能依靠 vn.py 独立开发策略并且进行实盘交易。



总体来说，vn.py 的设计理念可以概括为以下三点。

- **为交易而生 (By Traders, For Traders)**。相对于量化 IT 工程师追求低延时、高并发，交易员更加关注交易系统的灵活性和易用性，即能够快速地把交易思路转化成代码，即使遇到突发的行情也能快速修改代码，以捕捉稍纵即逝的赢利机会。
- **以量化投资机构的实盘交易为切入点**。完整的量化业务链包括数据获取、数据分析、策略回测和实盘交易。vn.py 是以实盘交易起家的，提供丰富的程序化交易接口，基本覆盖了国内外常规交易品种（如股票、期货、期权、外汇、数字货币）。其核心定位始终是实盘交易部分，提供包括算法交易（Algorithm Trading）、价差交易（Spread Trading）、期权交易、事前风控（Risk Manager）等，同时也提供了对于国内最常见的 CTA 类策略的支持：针对中高频单合约的 CtaStrategy 和针对中低频多合约的 TurtleStrategy。（TurtleStrategy 仅仅出现于 v1.9.1 以后的版本，该策略会在第 6 章专门介绍。）
- **开放的源代码**。闭源的商业软件尽管非常容易上手却存在安全性风险，而且仅提供一定程度的策略开发功能，不能修改底层的代码，因此不能满足使用者的所有需求。vn.py 开源的好处在于杜绝核心机密，支持使用者对底层接口和中层引擎进行修改，用户也可以根据自己的需要来添加新的上层应用。同时 vn.py 里面的代码有较为详细的注释，易于初学者读懂代码，大大降低了学习成本。开放的源代码和良好的框架为初学者提供了一条清晰的学习路径，学习 vn.py 有利于初学者慢慢地开发出自己的交易系统。

还有一点需要说明的是，相对于国内常见的商业量化软件，如 MultiCharts、TradeBlazer 和文华财经等，vn.py 在速度上有优势。下面分三个方面说明。

（1）**从驱动机制看**：vn.py 采用事件驱动，收到 Tick 后系统自身的处理逻辑很少，用户写的策略可以被立刻响应。

（2）**从软件功能看**：vn.py 没有自带 K 线显示（高 CPU 开销）和数据采集入库功能（高 I/O 开销），只有交易相关功能。

(3) 从编程语言看: MultiCharts、TradeBlazer 和文华财经使用的都是自身脚本, vn.py 基于 Python 语言, 而且如果会用 Cython 将核心逻辑转化到 C 语言, 则 vn.py 会快很多。

总的来说, 以上功能都有相应的代价, 比如没有自带 K 线显示, 则需要运维数据服务和使用配置文件等。

vn.py 开源项目因其卓越的设计理念, 在推出之后逐渐受到业内关注, 经项目负责人的不懈努力和 vn.py 社区的贡献, 使得 vn.py 在 GitHub 全球开源量化交易项目排名中始终位于前列, 特别在 Python 语言的实盘交易类型中稳居榜首。表 3-1 展示的是 vn.py 官方微信公众号(维恩的派 VNPIE)统计的 2018 年 6 月 GitHub 的 Star 排名。

表 3-1 GitHub 全球开源量化交易项目(2018 年 6 月 Star 排名)

排名	项 目	Star	上月	增加	语言	类型	介 绍
1	Zipline	7083	6626	457	Python	策略开发	Zipline, a Pythonic Algorithmic Trading Library
2	ccxt	6864	5236	1628	Python/Javascript/PHP	实盘交易	A JavaScript/ Python /PHP cryptocurrency trading library with support for more than 90 bitcoin/altcoin exchanges
3	Gekko	6659	6061	598	Javascript	实盘交易	A bitcoin trading bot written in node
4	vn.py	5733	5216	517	Python	实盘交易	基于 Python 的开源量化交易平台开发框架
5	TuShare	5720	5226	494	Python	数据工具	TuShare is a utility for crawling historical data of China stocks
6	Zenbot	4974	4556	418	Javascript	实盘交易	Zenbot is a command-line cryptocurrency trading bot using Node.js and MongoDB
7	Trump2Cash	4332	4225	107	Python	实盘交易	A stock trading bot powered by Trump tweets



续表

排名	项 目	Star	上月	增加	语言	类型	介 绍
8	RQAlpha	2618	2397	221	Python	策略开发	A extendable, replaceable Python algorithmic backtest && trading framework supporting multiple securities
9	Tribeca	2543	2306	237	Javascript	实盘交易	A high frequency,market making cryptocurrency trading platform in node.js
10	XChange	2295	2132	163	Java	实盘交易	XChange is a Java library providing a streamlined API for interacting with 60+Bitcoin and Altcoin exchanges providing a consistent interface for trading and accessing market data
						日期	2018-6-3

现在 vn.py 已经一步步地成长为一套全面的交易系统。2016 年, vn.py v1.0 发布, 增加了新的底层接口(飞创、飞鼠和 Oanda)、上层风控模块与行情记录模块; v1.4 版本增加了远程过程调用模块, 直达期货和盈透证券的交易接口, 以及增强 CTA 策略模块。2017 年, vn.py v1.5 发布, 增加了比特币交易接口和基于目标持仓来进行交易的策略模板, 随后的 v1.7 版本增加了价差交易模块。2018 年, vn.py v1.8 发布, 增加了 Web 界面、福汇和富途证券接口, 在期权方面增加了 Cython 版本的 Black-Scholes 期权定价模型。

随着数字货币的程序化交易越来越流行, 2018 年 8 月, 发布 vn.py v1.9.0 版本, 新增专门用于数字货币量化交易的 vn.crypto 模块, 目前提供针对数字货币交易所原生的 REST/WebSocket API 的高性能的 Python 接口封装设计, 以及适合 7×24 小时长时间交易需求的算法交易模块等。同时 vn.py 也开始从 Python2 版本升级到 Python3, 目前已经初步完成底层接口和中层引擎的兼容修改工作。

2018 年 12 月更新到 vn.py v1.9.1, 主要新增基于 CTA 策略模块实现的海龟信号单标的交易策略和实现完整的投资组合级别的海龟策略, 底层接口方面新增针



对 RESTful API 的统一客户端 RestClient, 针对 Websocket API 的统一客户端 WebsocketClient 等, 数据服务方面新增 CryptoCompare 的数字货币免费历史数据服务和 RqData 的期货、证券收费历史数据服务(这是目前推荐用于实盘的方案)。

同年 12 月末迎来 Python 2 的最终版本 v1.9.2 LTS, 这是第一个有长期支持的版本, vn.py 在 Python 2 上将不再有功能方面的更新, 主要新增了 TradeCopy 交易复制模块, 该模块主要是针对管理多个交易账户的用户而设计的; 更新了火币接口、OKEX 接口和富途接口; 新增图形化的 RQData 数据下载器; 新增 CccDataService 的日线数据下载功能(尽管本书使用 v1.9.0, 但是代码基本兼容 v1.9.2, 故读者也可以直接装 v1.9.2 版本)。

vn.py 已经正式开始了 Python 3 的升级计划, 预计发布的 v2.0 会基于 Python 3.7 进行代码重构, 尽可能使用 Python 3 新特性来实现。v2.0 会升级到 64 位作为主版本(官方编译的 API 不再支持 32 位), 而且实现独立的数据库对接模块, 兼容 MongoDB 和 SQL。

## 3.2 搭建 vn.py 运行环境

考虑到大部分初学者是 Windows 用户, 故 vn.py 环境的搭建是基于 Windows 的。

### 3.2.1 安装 Visual Studio 2013 社区版(特定版本)

Visual Studio(简称 VS)是微软公司开发的开发工具集, 它包括了整个软件生命周期所需要的大部分工具, 如 UML 工具、代码管控工具、集成开发环境(IDE)等。在 VS 中所写的目标代码适用于微软支持的所有平台。考虑到与 vn.py 兼容性的问题, 只能选择特定版本, 即 Visual Studio 2013 社区版, 其安装初始界面如图 3-1 所示。安装过程因为较简单, 只需一路单击“Next”按钮即可, 但安装时间有些长, 需要耐心等待。下面几节介绍的安装过程也是如此, 因此只列出



安装的初始界面。



图 3-1 Visual Studio 2013 社区版安装初始界面

### 3.2.2 安装代码编辑器工具：Sublime Text

Sublime Text 是一款轻量级的代码编辑器，支持多种编程语言的语法高亮显示，拥有优秀的代码自动完成功能，还可以将常用的代码片段保存起来，在需要时随时调用，其安装初始界面如图 3-2 所示。Sublime Text 可以用于查看和编辑 txt，json，Excel 和 Python 等文件。但是要开发比较大型的项目，或对某些代码进行深度解读，则需要用到更加强大的 Wing IDE。

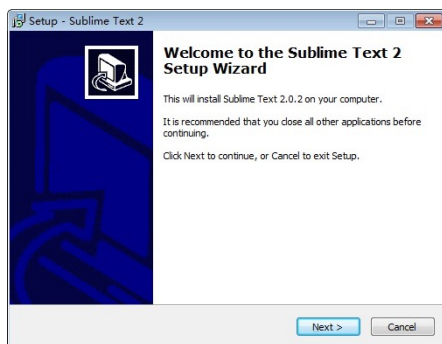




图 3-2 Sublime Text 安装初始界面

### 3.2.3 安装 Wing IDE

Wing IDE 是一个 Python 语言的 IDE，其中包括大量语法标签的高亮显示，可以提供调试 django 的功能，其安装初始界面如图 3-3 所示。Wing IDE 可以深度解构 vn.py 代码或者编辑管理 Python 的大型项目。



图 3-3 Wing IDE 安装初始界面

### 3.2.4 安装 MongoDB 数据库

MongoDB 是一个高性能、开源、无模式的文档型数据库，是当前 noSQL 数据库产品中最热门的一种。MongoDB 中一些概念和普通数据库不太一样，普通数据库有 database、table、row、field 的概念，在 MongoDB 里却依次叫 database、collection、document、field。它支持的数据结构非常松散，是类似 json 的 bson 格式，因此可以存储比较复杂的数据类型。MongoDB 最大的特点是它支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

推荐版本是 Windows 2008 plus( 64 位 )，支持 ssl 协议，如 mongodb-win32-x86\_



64-2008plus-ssl-4.0.0-signed, MongoDB 数据库安装初始界面如图 3-4 所示。



图 3-4 MongoDB 数据库安装初始界面

安装完成后，将 MongoDB 注册为 Windows 系统服务，确保其每次在开机后自动启动后台运营。

### 1. 创建数据库和日志目录

在 C 盘根目录下，按“Shift+鼠标右键”，打开命令提示窗口输入下面代码，如图 3-5 所示，生成两个文件夹：

```
md "\data\db" "\data\log"
```

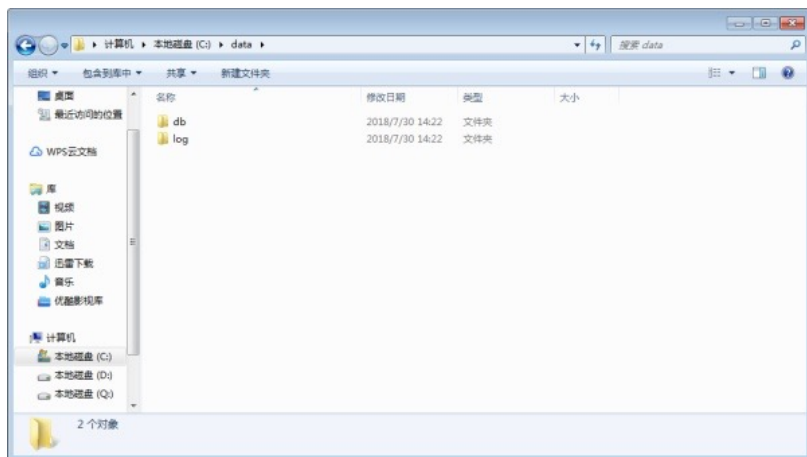


图 3-5 生成两个文件夹

(1) 启动 MongoDB 数据库。

在同一个命令提示窗口，输入下面命令：

```
"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="c:\data\db"
```

(2) 连接到 MongoDB。

在 C 盘根目录下新开命令提示窗口，输入下面命令，用于连接 MongoDB，成功连接后如图 3-6 所示。

```
"C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"
```



图 3-6 成功连接到 MongoDB

MongoDB 4.0 版本是默认开机启动的，不需要像以前的版本需要手动将 MongoDB 注册为 Windows 系统服务。重启系统后打开 Windows 任务管理器，可以发现已经成功在后台上启动，如图 3-7 所示。





图 3-7 MongoDB 后台启动

### 3.2.5 安装 Robo 3T

Robo 3T 是 MongoDB 可视化管理工具，其安装初始界面如图 3-8 所示。



图 3-8 Robo 3T 安装初始界面

安装完成后设置 localhost 为本地数据库，“27017”是默认的地址，如图 3-9 所示。

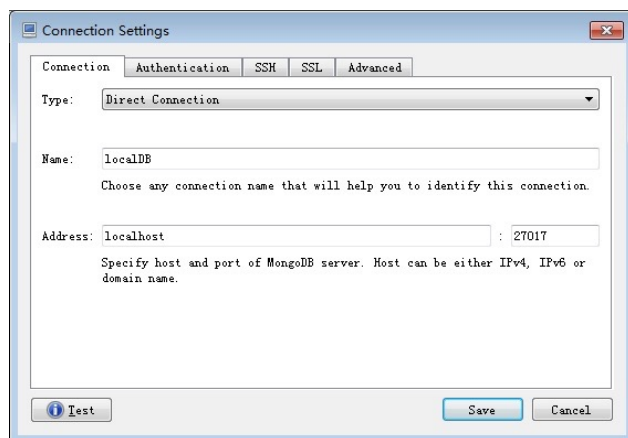


图 3-9 MongoDB 的默认地址

最后单击“Connect”按钮，就可以连接到 MongoDB 数据库，看到详细数据了，如图 3-10 所示。

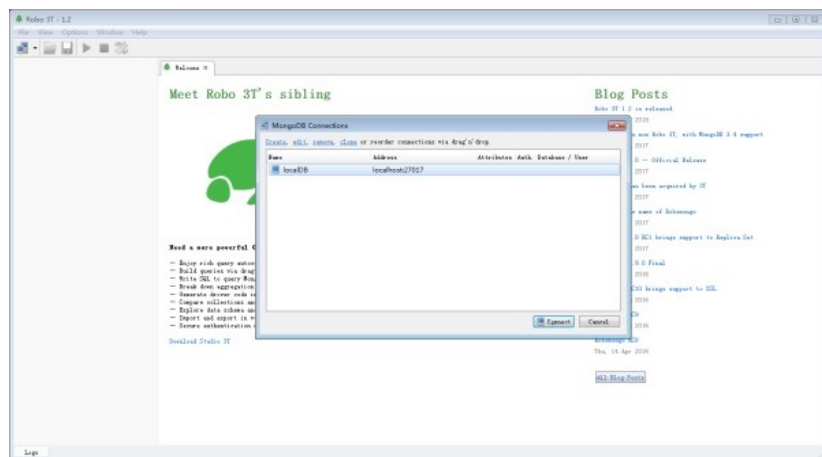


图 3-10 连接数据库

### 3.2.6 安装 vn.py

进入 GitHub 下载最新版本，Windows 用户选择第一个 zip 文件。当前笔者使用的版本是 v1.9.0，如图 3-11 所示。



## v1.9.0版本发布

vnpy released this 28 days ago

## Assets 2

[Source code \(zip\)](#)[Source code \(tar.gz\)](#)

图 3-11 vn.py 文件下载

下载好之后，解压，进入“C:\vnpy-1.9.0\vnpy-1.9.0”目录，按下“Shift+鼠标右键”，打开命令窗口，输入下面命令，将 vn.py 所用到的第三方库一键安装好，如图 3-12 所示。

```
install.bat
```

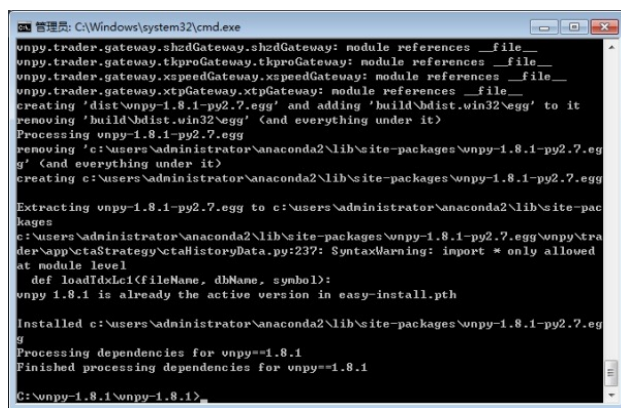


图 3-12 安装 vn.py 需要的第三方库

在同一个命令窗口，输入下面命令，目的是将 vn.py 安装进 Anaconda 的环境（若是 Anaconda2 安装在默认目录，则 vn.py 就在“C:\Users\Administrator\Anaconda2\Lib\site-packages”文件夹内），成功后在最后一行会显示“Finished processing dependencies for vnpy==1.9.0”。

```
python setup.py install
```

最后一步，打开 vn.py 图形应用程序 VnTrader。进入“C:\vnpy-1.9.0\vnpy-1.9.0\

examples\VnTrader”目录，按“Shift+鼠标右键”，打开命令窗口，输入下面命令，即可打开 VnTrader 图形交易界面。但是报错，显示缺少 queue 库，如图 3-13 所示（若是 vn.py1.9.2，则能够直接打开 VnTrader）。

```
python run.py
```

```
File "run.py", line 22, in <module>
    from vnpy.event import EventEngine
File "F:\ANACONDA\lib\site-packages\wnpy-1.9.0-py2.7.egg\wnpy\event\__init__.py", line 3, in <module>
    from .eventEngine import EventEngine, EventEngine2, Event, EVENT_TIMER
File "F:\ANACONDA\lib\site-packages\wnpy-1.9.0-py2.7.egg\wnpy\event\eventEngine.py", line 5, in <module>
    from queue import Queue, Empty
ImportError: No module named queue
```

图 3-13 vn.py 运行报错信息

这时不能用 pip install queue 安装，应该输入下面命令，安装 future 库，再运行 run.py（或者双击“VnTrader.bat”），成功后弹出 VnTrader 图形交易界面，如图 3-14 所示。

```
pip install future
```

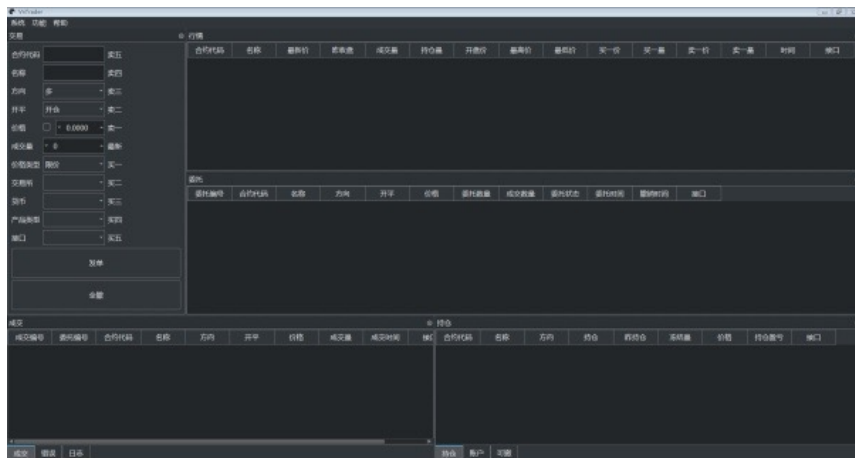


图 3-14 VnTrader 图形交易界面

### 3.2.7 更新 vn.py

更新 vn.py 时，需要在 Anaconda 内卸载旧版本的 vn.py，然后按照上述步骤



安装新版本即可，卸载命令如下：

```
pip uninstall vnpy
```

## 3.3 VnTrader 界面功能介绍

### 3.3.1 连接 CTP

首先在 SimNow 仿真交易网站上注册账号，具体操作是在“vnpy-1.9.0\examples\VnTrader”文件夹内用 Sublime Text 命令打开名为“CTP\_connection”的 json 文件，在其上分别输入下面信息。

- brokerID: 经纪商编号，在 SimNow 仿真交易环境默认是 9999。若是实盘交易账户，则需要询问开户所在的期货公司得到经纪商编号。
- mdAddress: 行情地址，分为 2 套。第 1 套模拟环境与实盘一致，只有在交易时段可以使用，比如 180.168.146.187:10011；第 2 套测试环境仅仅在非交易时段使用（交易日，16:00~次日 09:00；非交易日，16:00~次日 15:00），比如 180.168.146.187:10031。
- tdAddress: 交易地址，同样也分为 2 套。第 1 套 CTP 地址是 180.168.146.187:10001；第 2 套用于非交易时段的地址是 180.168.146.187:10030。
- userID: 用户编号，这是只有在注册成功后才能得到的 6 位纯数字编号。
- password: 用户密码，由用户自己定义。需要注意的是，第一次使用第 2 套 CTP 地址进行测试时，SimNow 会提示修改首次密码，否则无法连接 CTP。

在交易时段，用第 1 套 CTP 地址连接 CTP 接口，进行模拟盘测试的 json 文件配置 SimNow 账号信息如图 3-15 所示。



```
1 {  
2   "brokerID": "9999",  
3   "mdAddress": "tcp://180.168.146.187:10011",  
4   "tdAddress": "tcp://180.168.146.187:10001",  
5   "userID": "123456",  
6   "password": "123456789"  
7 }
```

图 3-15 SimNow 账号信息

Sublime Text 文本编辑器保存并退出后，打开 VnTrader 单击连接 CTP，连接成功后会在日志组件出现下面的 7 行字，如图 3-16 所示。

日志		
时间	内容	接口
09:25:39	交易合约信息获取完成	CTP
09:25:39	MongoDB连接成功	MAIN_ENGINE
09:25:39	结算信息确认完成	CTP
09:25:39	交易服务器登录完成	CTP
09:25:38	交易服务器连接成功	CTP
09:25:38	行情服务器登录完成	CTP
09:25:38	行情服务器连接成功	CTP

图 3-16 连接 CTP 成功显示信息

### 3.3.2 界面说明

整个 VnTrader 的窗口分为六大组件，如图 3-17 所示，下面介绍各组件功能。

① 交易组件：手动发送和撤销交易委托、订阅行情等。

② 行情组件：显示订阅实时行情的数据推送。

③ 委托组件：显示委托回报相关的数据推送。

④ 成交、日志与错误组件：显示成交相关的数据推送，调用底层接口时由于操作失败触发的错误信息推送，以及来自系统内各个组件发出的日志信息。

⑤ 持仓与资金组件：显示持仓数据信息和账户资金查询的数据结果。根据接口类型不同数据结果显示方式和频率也不同，有些接口通过定时轮询来实现，如



富途接口；有些接口则当状态只要发生变化就会推送新的信息，如 CTP 接口。

⑥ 菜单栏：主要实现 3 类功能，包括连接交易接口和 MongoDB 数据库；启动 CTA 策略模块、价差交易模块和风控管理模块；合约查询，编辑配置和还原窗口。

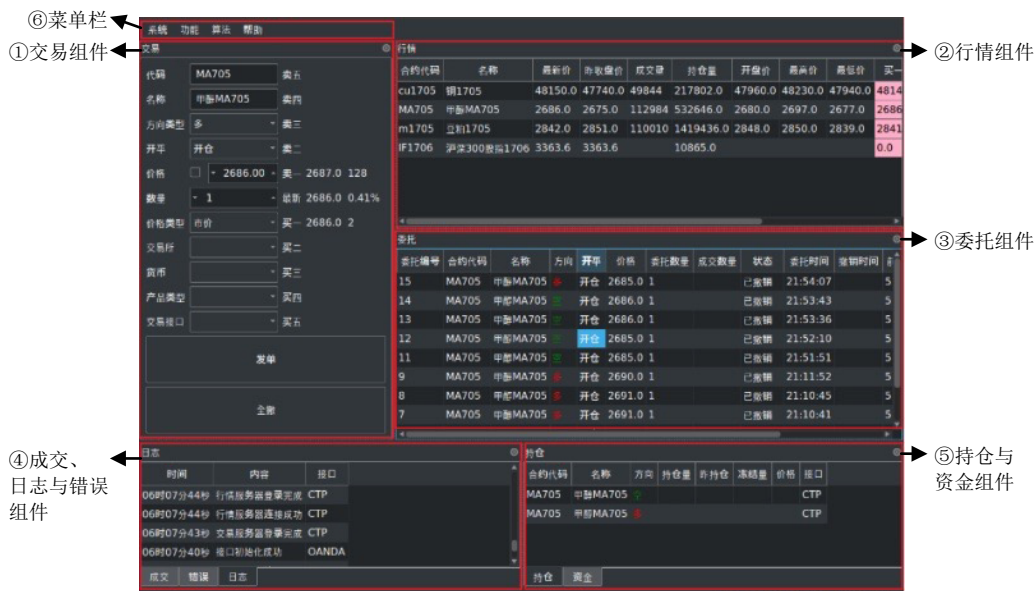


图 3-17 VnTrader 的六大组件

其中组件②③④⑤都是对当前行情和委托等信息的监控，如果通过程序化的方式自动交易，则交易的相关状态信息都会显示在这些组件中。需要手工干预时可以在窗口①进行操作，完成委托信息的输入、提交与撤单。

### 3.4 vn.py 架构

vn.py 的架构分为 3 层，分别是底层接口、中层引擎和上层应用，这 3 层应用模块相互影响，共同作用形成完整交易系统，如图 3-18 所示。

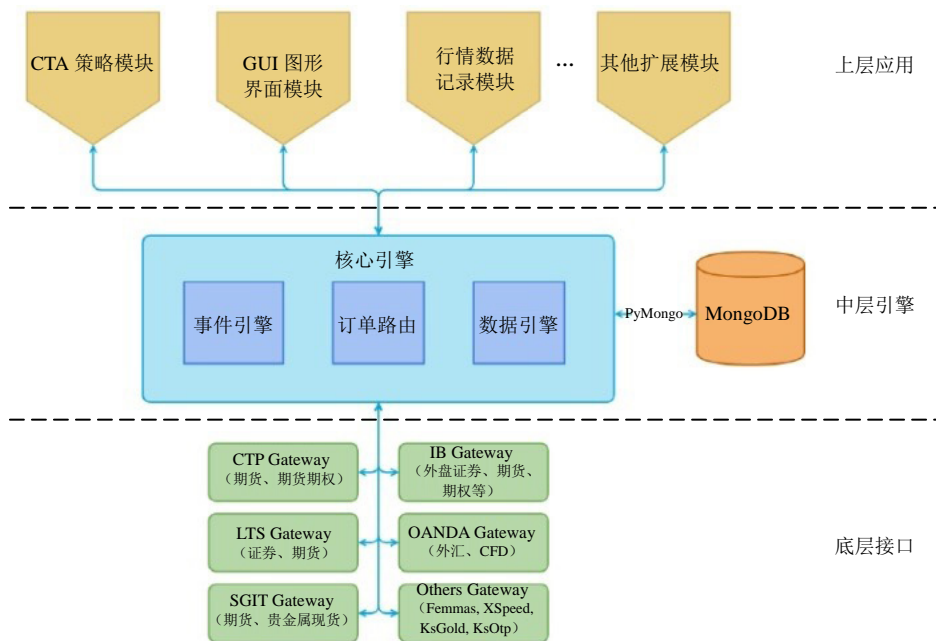


图 3-18 vn.py 的架构

### 3.4.1 底层接口

底层接口负责对接行情和交易 API，将数据推送到系统核心中，以及发送指令，比如下单、数据请求等。底层 API 接口处于文件夹“vnpy-1.8.1\vnpy\api”下，包含所有原生 C++/REST/WebSocket 的 API 接口，封装为 Python 交易接口的相关程序源代码及编译后的结果，其中包括：

- 上海中期的 CTP 接口（期货、期货期权）；
- 盈透证券的 IB 接口（外盘的证券、期货和期权等）；
- 华宝证券的 LTS 接口（证券、期货）；
- OANDA 接口（外汇、CFD）；
- 福汇接口（外汇、CFD）；
- 飞鼠的 SGIT 接口（期货、金交所的贵金属现货）；
- 大商所的飞创接口（期货、期货期权）；

- 中金所的飞马接口（期货、期货期权）；
- 金仕达黄金接口（金交所的贵金属现货）；
- 中泰证券的 XTP 接口（股票、债券、ETF、证券期权）；
- 富途证券接口（港股涡轮、美股）；
- BitMEX 接口（数字货币）；
- Bitfinex 接口（数字货币）；
- 其他 API 端口。

### 3.4.2 中层引擎

中层引擎包括事件引擎、订单路由和数据引擎，往下对接各种交易接口，往上服务于各种应用模块，它可以提供数据缓存、风险管理、订单路由等一系列量化交易中的通用功能。换句话说，中层引擎的功能是将程序中的各个组件，例如底层接口、数据库（MongoDB）接口等，整合到一个对象中，便于上层 GUI 图形界面调用。

中层引擎的具体工作流程如下：当底层接口接收行情推送后，行情数据会被推送到事件引擎上。事件引擎会在上层应用中检查哪个模块订阅了这个事件类型，并把行情数据推送到相应的应用模块中。当应用模块处理完行情数据后，直接调用主引擎的某些函数，再把对应的请求发送到底层接口中，如图 3-19 所示。

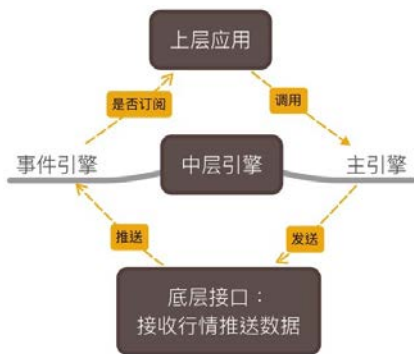


图 3-19 中层引擎的工作流程

事件引擎是 vn.py 的核心组件，也是大多数交易系统或回测引擎、甚至大多数交互程序的设计基础。相对于常见的批处理程序设计，事件驱动程序设计是另一种计算机程序设计模型，这种模型的程序运行流程是由用户的动作（如鼠标的按键，键盘的按键动作）或者其他程序的消息（如最新行情的推送）来决定的。在量化交易领域，事件引擎是为了解决 Python 全局锁 GIL 导致的多线程性能问题而开发的，通过单线程异步工作模式实现事件流的处理和分发，从而提高整个交易系统的性能。事件引擎在“vnpy-1.9.0\vnpy\event”文件夹中，其中 eventEngine.py 文件是事件引擎本身，eventType.py 文件用于定义事件类型。eventEngine.py 文件里定义了两个版本的事件引擎类，分别是 EventEngine 和 EventEngine2，它们的区别仅在于用不同的 Python 模块来实现。

### 3.4.3 上层应用

上层应用包括下面模块。

- **GUI 图形界面模块：**GUI 图形界面可以直观显示策略、行情、账户和交易的相关信息。vn.py 的图形界面 VnTrader 是基于 PyQt 开发的，主窗口主要包含账户相关窗口、行情窗口、交易窗口和错误及日志窗口等。最上方的菜单栏包括 3 个选项：“系统”选项用于连接交易 API 接口及 MongoDB 数据库；“功能”选项可以启动 CTA 策略模块、价差交易模块和风控管理模块；“帮助”选项可以进行合约查询、编辑配置和还原窗口。GUI 图形界面模块处于“vnpy-1.9.0\vnpy\trader”文件夹下，其中的“app”文件夹里包含官方实现的各种子模块，如 CTA 策略模块、价差交易模块等。
- **CTA 策略模块：**CTA 策略模块主要用于单标的的期货 CTA 策略和股票 T+0 日内交易策略，回测引擎和实盘引擎设计采用了完全兼容的 API 函数，用户可以使用同一套策略代码来实现回测研究和执行实盘交易。在扩展方面，基于 Python 生态丰富的机器学习库，交易员可以在策略代码中轻松实现各种机器学习的算法来提高效率。CTA 策略模块在“vnpy-1.9.0\vnpy\trader\app\ctaStrategy”文件夹中，其中包含数据定义 ctaBase、策略模板 ctaTemplate、回测引擎 ctaBacktesting、实盘引擎 ctaEngine、实盘配置 CTA\_setting，以及图形



应用 `uiCtaWidget`。

- **价差交易模块：**价差交易模块应用于多标的的统计套利策略，如跨市套利和跨品种套利。该模块支持任意数量的交易腿，可以根据各条腿的行情盘口实时计算价差的盘口价格和数量，也可以基于各条腿的持仓和成交数据，实现价差持仓管理。价差交易引擎也允许交易员自行开发各种 `Spreading` 算法，如狙击算法、做市算法等。价差交易模块在“`vnpy-1.8.1\vnpy\trader\app\spreadTrading`”文件夹中，其中包括数据定义 `stBase`、价差引擎 `stEngine`、价差算法 `stAlgo`、实盘配置 `CTA_setting`，以及图形应用 `uiStWidget`。
- **行情记录数据模块：**行情记录数据模块可通过相应的行情接口记录实盘 `Tick` 数据，并自动聚合为 `K` 线后插入 `MongoDB` 数据库，也支持无人值守的模式，每日定时自动更新行情数据到本地数据库中。行情记录数据模块在“`vnpy-1.9.0\vnpy\trader\app\dataRecorder`”文件夹中，包括数据定义 `drBase`、行情记录引擎 `drEngine`、实盘配置 `DR_setting`，以及图形应用 `uiDrWidget`。
- **远程过程调用模块：**远程过程调用（`RPC`）模块不仅支持远程函数调用，还支持服务端向客户端的数据广播推送。基于该模块，`vn.py` 相关的应用程序可以轻易实现前端数据展示和后端交易引擎间的解耦，即可在服务器上运行后端引擎，而在本机运行前端 `UI`，本机出现的软硬件故障不会影响程序后端的持续稳定运行。`RPC` 模块在“`vnpy-1.9.0\vnpy\trader\app\rpcService`”文件夹中，其中包括 `RPC` 引擎 `rsEngine`、`RPC` 客户端 `rsClient`、实盘配置 `RS_setting`，以及图形应用 `uiRsWidget`。
- **其他扩展模块。**

## 3.5 底层接口

### 3.5.1 CTP API 的工作原理

#### 1. CTP 介绍

CTP 是由上海期货交易所的上海期货信息技术有限公司专门为期货公司开发



的一套期货经纪业务管理系统，由交易系统、结算系统和风险控制系统三大系统组成。其中，交易系统主要负责订单处理、行情转发及银期转账业务；结算系统负责交易管理、账户管理、经纪人管理、资金管理、费率设置、日终结算、信息查询，以及报表管理等；风险控制系统则主要在盘中进行高速的实时试算，以及及时揭示并控制风险。CTP 能够同时连通国内四家期货交易所，支持国内商品期货和股指期货的交易结算业务，并能自动生成、报送保证金监控文件和反洗钱监控文件。

CTP 公开并对外开放交易系统接口，使用该接口可以接收交易所的行情数据和执行交易指令。该接口采用 API 的方式接入，使得用户可以随意开发自己的交易软件并直接连接到交易柜台上进行交易，在早期推动了国内量化交易的进程。CTP API 这一设计模式在期货领域已成为行业标准，例如飞马、飞创 Xspeed、华宝证券 LTS、金仕达黄金和恒生 UFT 等都采用类 CTP API 的设计。

## 2. API 功能介绍

《CTP 客户端开发指南》指出：CTP API 包含交易接口（Trader API）、风控接口（Risk API），以及结算接口（CSV）。使用这三个接口都可以实现与上期所技术综合交易平台系统的对接，从而进行交易、风险控制，以及对每日结算数据的保存。应该注意的是，风控接口和结算接口只供给期货公司内部使用，并不对投资者开放，故针对投资者的 CTP API 仅仅指交易接口。

交易接口主要用于获取交易所行情和下达交易指令，如订阅行情、下单、撤单、预埋单、银期转账、信息查询等。交易接口是三个接口中应用最广泛的接口，它的受众主要是：终端软件开发商（如快期）、对交易终端有特殊需求的个人、机构或自营单位投资者。

## 3. CTP API 文件

“vnpy-1.9.0\vnpy\api\ctp\ctpapi”文件夹内有 CTP API 原生文件，如图 3-20 所示。



文件名	详情
ThostFtdcTraderApi.h	C++头文件 包含交易相关的指令，如报单
ThostFtdcMdApi.h	C++头文件 包含获取行情相关的指令
ThostFtdcUserApiStruct.h	包含了所有用到的数据结构
ThostFtdcUserApiDataType.h	包含了所有用到的数据类型
thosttraderapi.dll	交易部分的动态链接库和静态链接库
thosttraderapi.lib	
thostmduserapi.dll	行情部分的动态链接库和静态链接库
thostmduserapi.lib	
error.dtd	包含所有可能的错误信息
error.xml	

图 3-20 CTP API 原生文件

注意，以.dll 和.lib 为后缀名的文件都是编译好的二进制文件，无法打开。所以从用户角度只需关注后缀名为.h 的文件。其中 TraderApi 定义交易相关的指令，MdApi 定义获取行情相关的指令，UserApiStruct 包含了 API 中用到的结构体的定义，UserApiDataType 包含了对 API 中用到的常量的定义。

## 4. API 通用规则

### (1) 命名规则

API 命名规则如图 3-21 所示。

消息	格式	示例
请求	Req—	ReqUserLogin
响应	OnRsp—	OnRspUserLogin
查询	ReqQry—	ReqQryInstrument
查询请求的响应	OnRspQry—	OnRspQryInstrument
回报	OnRtn—	OnRtnOrder
错误回报	OnErrRtn—	OnErrRtnOrderInsert

图 3-21 API 命名规则



## (2) API 分类

API 分为两种，如图 3-22 所示。

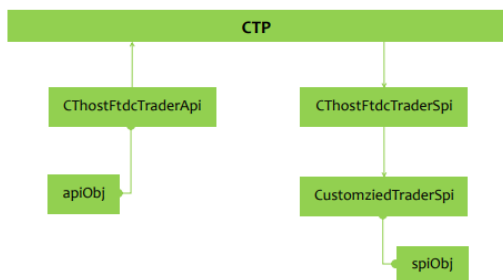


图 3-22 API 分类

- Spi 类（如 ThostFtdcTraderApi.h 的 CThostFtdcTraderSpi 类）包含所有的响应和回报函数，用于接收 CTP 发送的信息。开发者需要继承该接口类，并实现其中相应的虚函数。
- Api 类（如 ThostFtdcTraderApi.h 的 CThostFtdcTraderApi 类）包含主动发起请求和订阅的接口函数，开发者直接调用即可。

## (3) 通用参数

- nRequestID：客户端发送请求时要为该请求指定一个请求编号。交易接口会在响应或回报中返回与该请求相同的请求编号。当客户端频繁操作时，很有可能会造成同一个响应函数被调用多次，在这种情况下，能将请求与响应关联起来的纽带就是请求编号。
- IsLast：当响应函数需要携带的数据包过大时，该数据包会被分割成数个小的数据包并按顺序逐次发送，在这种情况下同一个响应函数会被调用多次，而参数 IsLast 就是用于描述当前收到的响应数据包是不是所有数据包中的最后一个。
- RspInfo：该参数用于描述请求执行过程中是否出现错误。该数据结构中的属性 ErrorID 如果是 0，则说明该请求被交易核心认可通过，否则，该参数将描述交易核心返回的错误信息。



- error.xml: 此文件中包含所有可能的错误信息。

#### (4) 接口初始化

通过创建和初始化行情接口和交易接口，开启交易接口的工作流程有以下步骤：

a) 创建继承自 SPI，并创建出实例，以及 API 实例。

- SPI 指 CThostFtdcTraderSpi 或 CThostFtdcMdSpi。
- API 指 CThostFtdcMdApi 或 CThostFtdcTraderApi。

b) 向 API 实例注册 SPI 实例。

c) 向 API 实例注册前置地址。交易接口需要注册交易的前置地址，行情接口需要注册行情的前置地址。

d) 订阅公有流（仅限交易接口，行情接口不需要）。用于接收公有数据，如合约在场上的交易状态。默认模式是从上次断开连接处继续收取交易所发布的数据（Resume 模式），开发者还可以指定全部重新获取（Restart），或登录后获取（Quick）。

e) 订阅私有流（仅限交易接口，行情接口不需要）。用于接收私有数据，如报单回报。默认模式是从上次连接断开处继续收取交易所发布的数据（Resume 模式），开发者还可以指定全部重新获取（Restart），或登录后获取（Quick）。

f) 初始化（Init）。

g) 等待线程退出（Join）。

#### (5) 行情接口的工作原理

简单展示一下行情接口的工作原理，如图 3-23 所示，让初学者对 API 的工作原理有直观的印象。



图 3-23 行情接口的工作原理

## 3.5.2 CTP API 的 Python 封装设计

### 1. Python 封装的必要性

基于 C++ 的 CTP API 只有先进行 Python 封装，才能直接被 vn.py 调用。在 ThostFtdcUserApiStruct.h 中定义的结构体，既无法在 Python 环境中直接创建，也无法提取结构体中包含的数据（回调函数）。Python 虚拟机是基于 C 语言实现的，Python 的对象在 C 环境中对应的是 PyObject 对象，故 C++ 也无法识别 Python 直接传递的参数。

### 2. 封装设计思路

原生 API 中的每个功能分为了两个类，分别是：包含回调函数的 Spi 类和包含主动函数的 Api 类，这种设计能让用户更好地分清不同的功能。但是从面向对象编程的角度出发，把两个类封装到一起更为方便。因此，在 Python 的 API 中，会把 Spi 和 Api 两个类的功能封装到一个类中。

原生 API 中的回调函数被触发后必须快速返回，否则会导致其他数据的推送

被阻塞，阻塞时间长了还有可能导致 API 崩溃，因此回调函数中不适合包含耗时较长的计算逻辑。例如，某个 Tick 行情推送后，如果用户在回调函数中写了一些比较复杂的计算，如循环计算，耗时超过 3 秒钟，则在这个 3 秒钟中，用户是收不到其他行情推送的，且很可能 3 秒钟后会出现 API 崩溃。这里的解决方案是使用生产者-消费者模型，在 API 中包含一个缓冲队列，当回调函数收到新的数据信息时只是简单存入缓冲队列中并立即返回，而数据信息的处理及向 Python 中的推送则由另一个工作线程来执行。

基于 C++ 的 CTP API 的函数中使用了大量的结构体用于数据传送，若所有结构体都要封装成对应的 Python 类，工作量太大也非常容易出错。解决方案是使用 Python 中的 Dict 字典。字典有着更为高级的数据结构，它本质是一个哈希表，同一个字典内键和值的类型允许不同，这个特性使得字典可以非常方便地用来代替 C++ 的结构体。

原生 API 的函数名开头都是大写字母，为了便于分辨及符合 Python 的 PEP8 编码规则，Python 封装后的函数都以小写字母开头。例如：原生 API 中以 On 开头的回调函数（如 OnRspUserLogin）对应的 Python API 的回调函数直接改为以 on 开头的函数（如 onRspUserLogin）；主动函数（如 ReqUserLogin）对应的封装后 API 中的主动函数改为首字母小写（如 reqUserLogin）。

### 3. 封装后 API 工作流程

#### （1）主动函数

- 用户在 Python 程序中调用封装 API 的主动函数，并直接传入 Python 变量（PyObject 对象）作为参数。
- 封装 API 将 Python 变量转换成 C++ 变量。
- 封装 API 调用原生 API 的主动函数，并传入 C++ 变量作为参数。

#### （2）回调函数

- 交易柜台通过原生 API 的 C++ 回调函数推送数据信息，传入参数为 C++ 变量。

- 封装 API 将 C++ 变量转换为 Python 变量。
- 封装 API 调用封装后的回调函数向用户的 Python 程序中推送数据，并传入 Python 变量作为参数。

### 3.5.3 CTP API 对接中层引擎原理

当原生 C++ 的 CTP API 封装成功后，就可以对接到中层引擎，对接流程分为两步：

- (1) 将 API 的回调函数收到的数据推送到程序的中层引擎中，等待处理；
- (2) 将 API 的主动函数进行一定的简化封装，便于中层引擎调用。

下面通过案例代码来具体展示 CTP API 对接中层引擎的具体流程。

#### 1. 回调函数

通过回调函数收到 API 的数据推送后，创建不同类型的 Event 对象（来自事件驱动引擎模块），在事件对象的数据字典 dict\_ 中保存需要具体推送的数据，然后推送到事件驱动引擎中，由其负责处理。

在回调函数收到的数据中，data 和 error 分别对应的是保存主要数据（如行情）和错误信息的字典，n 是该回调函数对应的请求号（即调用主动函数时的 reqid），last 是一个布尔值，代表是否为该次调用的最后返回信息。

我们主要对 data 字典感兴趣，因此选择在事件中整体推送。

而 error 字典每次收到后应当立即检查是否包含错误信息（因为即使没有发生错误也会推送），若有则自动保存为一个日志事件（通过日志监控控件显示出来）。

服务器连接完成（onFrontConnected）后，检查是否已经填入了用户名等登录信息，若有则自动登录（请参考后面主动函数中的示例）。

登录完成（onRspUserLogin）后，自动订阅\_setSubscribed 中之前已经订阅过



的合约。

收到行情推送（onRtnDepthMarketData）后，我们选择创建两种事件：一种是常规行情事件（通常适用于市场行情监控 GUI 等对所有行情推送都关注的组件）；另一种是特定合约行情事件（通常适用于算法等仅关注特定合约行情的组件）。

当调用有返回信息的主动函数时，需要传入本次请求的编号，此时先将 reqid 加 1，再作为参数传入主动函数中。

其示例代码如下：

```
#-----
def onFrontConnected(self):
    """服务器连接"""
    event = Event(type_=EVENT_LOG)
    event.dict_['log'] = u'行情服务器连接成功'
    self.__eventEngine.put(event)

    # 如果用户已经填入了用户名等，则尝试自动登录。
    if self.__userid:
        req = {}
        req['UserID'] = self.__userid
        req['Password'] = self.__password
        req['BrokerID'] = self.__brokerid
        self.__reqid = self.__reqid + 1
        self.reqUserLogin(req, self.__reqid)

.....

#-----
def onRspUserLogin(self, data, error, n, last):
    """登录回报"""
    event = Event(type_=EVENT_LOG)

    if error['ErrorID'] == 0:
        log = u'行情服务器登录成功'
    else:
        log = u'登录回报，错误代码: ' + unicode(error['ErrorID']) + u', ' + u'错误信息: ' + error['ErrorMsg'].decode('gbk')
```

```

event.dict_['log'] = log
self.__eventEngine.put(event)

# 重连后自动订阅之前已经订阅过的合约
if self.__setSubscribed:
    for instrument in self.__setSubscribed:
        self.subscribe(instrument[0], instrument[1])

.....

#-----
def onRtnDepthMarketData(self, data):
    """行情推送"""
    # 行情推送收到后, 同时触发常规行情事件, 以及特定合约行情事件, 用于满足不同类型的监听。

    # 常规行情事件
    event1 = Event(type_=EVENT_MARKETDATA)
    event1.dict_['data'] = data
    self.__eventEngine.put(event1)

    # 特定合约行情事件
    event2 = Event(type_=(EVENT_MARKETDATA_CONTRACT+data['InstrumentID']))
    event2.dict_['data'] = data
    self.__eventEngine.put(event2)

...

```

## 2. 主动函数

主动函数仅封装了两个功能：登录（login）和订阅合约（subscribe）。

对于登录函数而言，传入参数包括服务器前置机地址 address、用户名 userid、密码 password，以及经纪商代码 brokerid。函数调用后，先将 userid，password 和 brokerid 保存下来，然后注册服务器地址 registerFront，并初始化连接。连接完成后，onFrontConnected 回调函数会被自动调用，之后发生的操作请参考回调函数工作流程。

订阅合约方面，CTP API 在期货方只需要传入合约代码，但是证券类的接口需要同时传入合约的代码，以及合约所在的交易所（因为存在两个证券交易所相同代码的情况）。发送订阅请求后，将该订阅请求保存在 setSubscribed 集合中，



使得断开重连时可以自动重新订阅。

其示例代码如下：

```
#-----
def login(self, address, userid, password, brokerid):
    """连接服务器"""
    self.__userid = userid
    self.__password = password
    self.__brokerid = brokerid

    # 注册服务器地址
    self.registerFront(address)

    # 初始化连接成功后会调用 onFrontConnected
    self.init()

#-----
def subscribe(self, instrumentid, exchangeid):
    """订阅合约"""
    req = {}
    req['InstrumentID'] = instrumentid
    req['ExchangeID'] = exchangeid
    self.subscribeMarketData(req)

    instrument = (instrumentid, exchangeid)
    self.__setSubscribed.add(instrument)
```

## 3.6 事件引擎

事件引擎是 vn.py 的核心组件，通过对接底层接口和上层应用模块，维持整个交易系统的正常运作。与常见的时间驱动不同，事件引擎的设计基于事件驱动。

### 3.6.1 时间驱动

时间驱动就是让计算机每隔一段时间固定运行一次脚本，脚本自身可以很长、包含非常多的步骤，但是这种程序的运行机制相对比较简单、容易理解。下面提



供时间驱动在量化交易领域应用的 2 个例子。

例子 1：每隔 2 分钟，获取交易账户的最新净值、持仓信息、委托状态等。

例子 2：每隔 1 秒钟，检查一次最新收到的股指期货 Tick 数据，更新 K 线和其他技术指标，检查是否满足趋势策略的下单条件，若满足则执行下单。

但是对于速度要求较高的量化交易（如日内 CTA 策略、高频策略等），时间驱动程序却存在一个非常大的缺点：对数据信息在反应操作上的处理延时。在例子 2 中，在每次逻辑脚本运行完等待的那 1 秒钟里，程序对于接收的新数据信息（行情、成交推送等）是不会做出任何反应的，只有在等待时间结束后脚本再次运行时才会进行相关的计算处理，导致交易成本提高。

若是缩小时间驱动程序的等待间隔，提高程序运行的频率。比如在例子 2 中，每隔 100 毫秒检查获取最新股指期货 Tick 数据，尽管可以在一定程度降低滑点费用，却会导致 CPU 计算资源的浪费，大大降低计算机运转性能。

## 3.6.2 事件驱动

所谓事件驱动，简单地说就是你点什么按钮（即产生什么事件），计算机执行什么操作（即调用什么函数）。当然事件不仅限于用户的操作。事件驱动的核心自然是“事件”。从事件角度说，事件驱动程序的基本结构包括事件收集器、事件发送器和事件处理器。事件收集器专门负责收集所有事件，包括来自用户的（如鼠标、键盘事件等）、硬件的（如时钟事件等）和软件的（如操作系统、应用程序本身等）。事件发送器负责将收集器收集到的事件分发到目标对象中。事件处理器做具体的事件响应工作，它往往要到实现阶段才完全确定，因而需要运用虚函数机制。

回到交易系统的设计，事件驱动就是当某个新的事件被推送到程序中时（如 API 推送新的行情、成交等），程序立即调用和这个事件相对应的处理函数进行相关的操作。若没有事件推送过去，该程序是不会被触发的，这样可以完美地解



决在上面的例子 2 中 CPU 资源浪费的问题：交易程序对股指期货 Tick 数据进行监听，当没有新的行情过来时，程序保持监听状态不进行任何操作；当收到新的数据时，数据处理函数立即更新 K 线和其他技术指标，并检查是否满足趋势策略的下单条件，据此执行下单。

事件驱动设计的另一个优点就是增强交易系统的扩展性，因为事件驱动引擎可以管理不同事件的事件监听函数并执行所有和事件驱动相关的操作。例如收到 CTP API 推送的最新股指期货 Tick 数据，针对这个数据，系统需要进行如下处理：

- 更新行情监控表中股指期货的行情数据（表格更新）。
- 各个策略均需要运行一次内部算法，检查该数据是否会触发策略执行下单（运算、下单）。
- 风控系统需要检查最新行情价格是否会导致账户的整体风险超限，若超限需要进行报警（运算、报警）。

### 3.6.3 事件引擎工作流程

图 3-24 是由 vn.py 社区贡献的事件引擎工作流程图。

事件驱动流程可以分为两个方面：一方面是用户在主动订阅端通过事件引擎把订阅请求发送到 API 接口；另一方面是回调推送端把最新的 Tick 行情数据由事件引擎推送到具体的交易策略中进行运算。

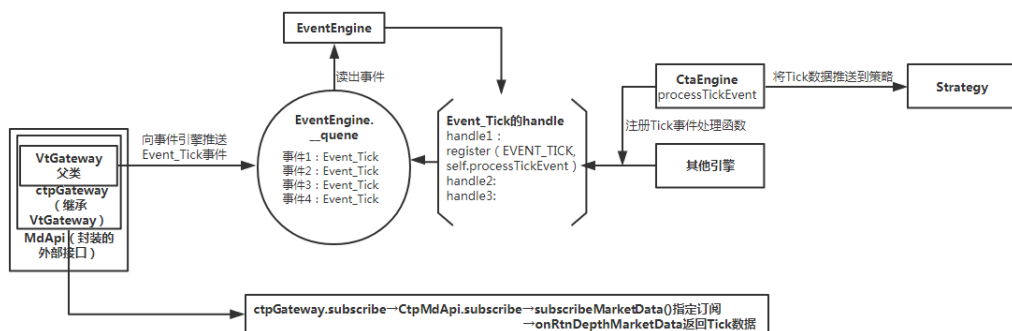


图 3-24 事件引擎工作流程图

### 1. 主动订阅端

- 用户发起调用 `mainEngine.subscribe` 函数。
- `mainEngine.subscribe` 中调用 `ctpGateway.subscribe` 函数。
- `ctpGateway.subscribe` 中调用 `ctpMdApi.subscribe` 函数。
- `ctpMdApi.subscribe` 中调用 C++封装的 `MdApi.subscribeMarketData` 函数，将订阅行情的请求最终通过底层 C++ CTP API 发出。

### 2. 回调推送端

- `ctaEngine` 对象向 `eventEngine` 中注册 `EVENT_TICK` 事件类型的处理函数句柄 `ctaEngine.processTickEvent`。
- C++ CTP API 收到 Tick 推送，自动回调 `MdApi.onRtnDepthMarketData` 函数推送行情数据字典 `data`。
- `MdApi.onRtnDepthMarketData` 中将 `data` 里的数据读取并转化成 `VtTickData` 对象，并调用 `ctpGateway.onTick` 函数。
- `ctpGateway.onTick` 函数将 `VtTickData` 对象包装成类型为 `EVENT_TICK` 的行情事件对象 `Event`，并调用 `eventEngine.put` 函数，放入事件引擎的缓冲队列。
- 事件引擎的工作线程，从缓冲队列中读取最新的行情事件后，根据 `EVENT_TICK` 事件类型去查找缓存在内部字典中的处理函数列表，并将事件对象作为入参，遍历调用到列表中的处理函数 `ctaEngine.processTickEvent`。
- `ctaEngine.processTickEvent` 查看 Tick 的代码 `vtSymbol`，并调用交易该代码合约的策略对象 `strategy.onTick` 函数，最终去运行策略中的逻辑。

## 3.6.4 事件引擎结构

事件引擎处于“vnpy-1.9.0\vnpy\event”文件夹中的 `eventEngine.py` 内，打开该文件可以发现有两个版本的事件驱动引擎，两个版本的区别仅在于定时器是通过不同的 Python 模块实现的，`EventEngine` 直接用 `QTimer`，而 `EventEngine2` 用线程实现定时器。



整体来说，事件引擎主要由以下 5 部分组成：

- 事件队列(`__queue`)。
- 事件处理线程(`__thread`)。
- 事件处理函数字典(`__handlers`) / 通用事件处理函数列表(`__generalHandlers`)。
- 定时器(`__timer`)。
- 引擎开关(`__active`)。

### 1. 事件队列

事件队列(`__queue`)是一个队列(`Queue`)对象。通过事件引擎的 `put(event)` 方法，用户可向事件队列中传入事件(`Event`)对象，事件引擎开始运行后会不断尝试从队列中取出事件对象、从事件处理函数字典中寻找对应该事件类型的一个或多个事件处理函数并依次调用之。引擎调用监听对应事件类型的事件处理函数的过程，就是引擎被事件“驱动”的过程。

### 2. 事件处理线程

事件处理线程(`__thread`)是一个线程(`Thread`)对象，该线程中运行的是 `__run()` 函数，即对事件队列的轮询及事件处理函数的调用都运行在该线程中。将事件处理放到一个单独的线程中是必要的，最重要的原因是当创建事件引擎并调用 `start()` 函数将之启动的时候，函数可以及时返回而保持事件引擎的连续运行，不会因为事件引擎的运行而阻塞。

### 3. 事件处理函数字典/通用事件处理函数列表

事件处理函数字典(`__handlers`)是一个字典(`dict`)，用于存储事件类型和事件处理函数的监听关系。事件处理函数字典的 `Key` 一般是事先定义好的常量，用来表示单一的事件类型，字典的 `Value` 是列表(`list`)，用来存储处理对应事件类型的处理函数（即一个事件类型可以被多个事件处理函数监听）。用户可以通过 `register(type_, handler)`和 `unregister(type_, handler)`这两个函数向事件处理字典中对应事件类型的函数列表中添加或移除事件处理函数，即注册或注销事件处理

函数。有关事件类和事件处理函数将会在后面详细说明。

通用事件处理函数（`__generalHandlers`）是一个列表（`list`），用于存储一系列监听所有事件的函数。用户可以通过 `registerGeneralHandler(handler)` 和 `unregisterGeneralHandler(handler)` 这两个函数注册或注销通用事件处理函数。

#### 4. 定时器

定时器（`__timer`）是一个运行在单独线程中的函数（`__runTimer()`），在定时器开关（`__timerActive`）打开后，该函数会定期（间隔 `__timerSleep` 秒）向事件队列中插入一个类型为 `EVENT_TIMER` 的事件。用户可以通过注册对应该事件类型的事件处理函数，实现函数的定期运行。如 `VnTrader` 便是通过注册了监听 `EVENT_TIMER` 的事件处理函数实现了对持仓和账户信息的定期查询和实时更新。

#### 5. 引擎开关

引擎开关（`__active`）是一个布尔型（`bool`）变量，只有当该变量值为 `True` 的时候事件引擎才会开始轮询，用户通过引擎的 `start()` 和 `stop()` 函数修改该变量的值来实现控制事件引擎的启动和终止。

## 3.7 上层应用

### 3.7.1 PyQt 介绍

开发图形界面的需求产生自实时监控交易策略执行情况，因为交易员很难监控在 `Shell` 界面上不断更新的日志信息，出现问题后也不易排查问题。而且随着国内越来越多衍生品的推出，很多新型的交易策略从全自动转向了半自动，经常需要交易员的手动干预，例如暂停策略、盘中微调参数，以及投资组合层面的风险管理等。

`vn.py` 的 GUI 界面是基于 `PyQt` 开发的。`PyQt` 是一个用于创建 GUI 应用程序



的跨平台的工具包，它将 Python 编程语言和 Qt 库成功地融合在一起。其中 Qt 库是基于 C++ 开发的，具有高性能的特点。PyQt 的核心是 Qt 库，但是在使用方式保留了 Python 优雅的语法和快速开发的能力。

PyQt 包含了大约 440 个类型、超过 6000 个的函数和方法。其中 QtGui 模块包含了大多数的 GUI 类型，包含按钮、文本框、列表等常见控件，还包含了基于 MVC 设计模式的列表、表格、树形控件，同时还提供了一个能够容纳成千上万个元素的画布控件，它们可以放置各种控件和图形。

### 3.7.2 GUI 组件构成

从功能上看，所有交易平台的 GUI 组件都可以分为 3 类，分别是数据监控（被动）、功能调用（主动），以及混合类。

#### 1. 数据监控

行情监控组件用于监控实时行情数据，每当 API 端有新的行情数据推送时立即进行更新。vn.py 的图形界面 VnTrader 中的合约查询选项就属于数据监控组件，如图 3-25 所示。



合约代码	交易所	vt系统代码	名称	产品类型	合约大小	最小价格变动	标的代码
zn1910	SHFE	zn1910	锌1910	期货	5	5.0	
zn1909	SHFE	zn1909	锌1909	期货	5	5.0	
zn1908	SHFE	zn1908	锌1908	期货	5	5.0	
zn1907	SHFE	zn1907	锌1907	期货	5	5.0	
zn1906	SHFE	zn1906	锌1906	期货	5	5.0	
zn1905	SHFE	zn1905	锌1905	期货	5	5.0	
zn1904	SHFE	zn1904	锌1904	期货	5	5.0	
zn1903	SHFE	zn1903	锌1903	期货	5	5.0	
zn1902	SHFE	zn1902	锌1902	期货	5	5.0	
zn1901	SHFE	zn1901	锌1901	期货	5	5.0	
zn1812	SHFE	zn1812	锌1812	期货	5	5.0	
zn1811	SHFE	zn1811	锌1811	期货	5	5.0	
zn1810	SHFE	zn1810	锌1810	期货	5	5.0	
zn1809	SHFE	zn1809	锌1809	期货	5	5.0	
zn1808	SHFE	zn1808	锌1808	期货	5	5.0	
zn1807	SHFE	zn1807	锌1807	期货	5	5.0	
wr1910	SHFE	wr1910	橡胶1910	期货	10	1.0	

图 3-25 数据监控组件

数据监控组件主要用于对交易平台中的各项数据实现实时更新或者手动更新的显示监控，最常用的包括行情、报单、成交、持仓和日志等。监控组件中最常见的类型是表格，对应 PyQt 中的 `QTableWidget` 组件，表格中的单元格则使用 `QTableWidgetItem` 组件。

不同的监控内容需要实现不同的数据更新方法，例如日志、成交类数据应该使用插入更新（即每条新的数据都应该插入新的一行），行情数据应该使用固定位置更新（即在表格中固定的单元格位置更新数据），以及主要针对持仓和报单数据的混合更新（即已经存在的数据直接在对应的位置更新，否则插入新的一行）。

## 2. 功能调用

用户用功能调用组件在主动订阅端发起请求，通过中层引擎传递到底层接口中。在 VnTrader 的连接交易接口（如连接 CTP，飞马）选项，就是把 json 文件中的用户名、密码、服务器地址等参数传递到底层接口进行登录，如图 3-26 所示。



图 3-26 功能调用组件

主动调用组件通常在工作原理上较为简单，用户只需在界面上放置所需的组件（按钮、下拉框等），并将组件的信号和中层引擎暴露的功能函数连接上即可。

## 3. 混合类

混合类指的是该组件既包含数据回调推送功能，也包括主动订阅功能。在 VnTrader 界面上，位于左侧部分的交易组件用于填入下单参数后调用中层引擎的



下单功能发单, 位于右侧的行情组件则用于监控用户输入的合约代码的实时行情, 如图 3-27 所示。



图 3-27 混合类组件



# 第 4 章

## 在 vn.py 中实现 CTA 策略

本章首先介绍 vn.py 提供的数据库解决方案，然后介绍用于生成具体 CTA 策略的相关支持模块，如 K 线生成、K 线管理和策略模板，最后讲述回测和优化模块。

### 4.1 数据库解决方案

vn.py 提供两套数据库解决方案：第一套是 CSV ( Comma Separate Values ) 加载模块，用于载入几个月甚至几年的历史数据进行策略回测；第二套是数据库下载模块，用于下载当前几天的历史数据，在 VnTrader 中启动策略，进行实盘或模拟盘交易。

#### 4.1.1 CSV 加载模块

相对于其他闭源商业软件提供长期数据库服务，vn.py 的解决方案是从第三方下载历史数据，以 CSV 格式插入 MongoDB 数据库，进行以后的策略回测。但是从第三方下载数据库的缺点是所下载的数据格式都略有差别，必须转换成统一的数据格式插入数据库中，因此用户需要自行开发针对特定数据源的 CSV 加载模块。目前，vn.py 官方已实现对常用数据库源 CSV 格式数据的转化，包括：



- MultiCharts。
- TradeBlazer。
- TB 极速版。
- 中银国际证券通达信。
- OKEX。

当用户从第三方数据源下载 csv/txt 数据时，可以参考上面的模板，把数据转换为统一格式并插入 MongoDB 数据库中。CSV 加载模块处于“vnpy-1.9.0\vnpy\trader\app\ctaStrateg”文件夹下的 ctaHistoryData.py 文件内，下面以 MultiCharts 导出的 CSV 文件为例，具体讲解 CSV 加载模块的运作过程。

### 1. vtConstant 和 ctaBase 用于定义默认空值和数据库名称

在默认空值中定义 4 种空值分类，包括字符串、中文字段、整数和浮点数。

MongoDB 数据库定义 3 种级别数据，具体如下。

- DAILY\_DB\_NAME: MongoDB 数据库保存日 K 线级别数据。
- MINUTE\_DB\_NAME: MongoDB 数据库保存 1 分钟 K 线级别数据。
- TICK\_DB\_NAME: MongoDB 数据库保存 Tick 级别数据。

日 K 线级别数据及 1 分钟 K 线级别数据是相对容易获取的，如 MultiCharts、TradeBlazer 的免费账号可以下载到 3 年左右的 1 分钟 K 线级别数据，但是要获得更长周期的 1 分钟 K 线级别数据或者 Tick 级别数据，就需要升级到年费用户。RQData 期货终端也有 7 天试用账号可以免费下载期货数据。

Tick 级别数据对网络延时的要求非常高。尽管 vn.py 的 DataRecording 模块支持用户自动记录 Tick 级别数据，然后插入 MongoDB 数据库中，但是 Tick 级别数据质量并不好，这种做法并不推荐。另外一种做法是购买大型数据提供商提供的的数据，如通联数据和万得的数据，它们应对网络延时的方法是三网（电信、移动、联通）同步记录数据，而且在盘后会有专门人员进行数据清洗。还有一种最昂贵和精准的解决方法——直接向交易所购买数据。交易所数据是原始的、未经清洗

的。因为第三方数据提供商在清洗数据的过程中可能会无意间把隐含赚钱机会的数据也清洗走了。

CTA 策略作为量化交易入门首选的中低频策略，持仓周期可以是几分钟到几天，并不需要 Tick 级别数据，用 1 分钟 K 线级别数据进行策略回测即可。

下面是示例代码。

```
# 默认空值
EMPTY_STRING = ''
EMPTY_UNICODE = u''
EMPTY_INT = 0
EMPTY_FLOAT = 0.0

# 数据库名称
TICK_DB_NAME = 'VnTrader_Tick_Db'
DAILY_DB_NAME = 'VnTrader_Daily_Db'
MINUTE_DB_NAME = 'VnTrader_1Min_Db'
```

## 2. VtBarData 用于规定 K 线的格式

VtBarData 继承父类 VtBaseData，交易所、品种代码和 vt 系统代码的数据类型是字符串，K 线的高开低收（OHLC）数据类型是浮点数，日期和时间的数据类型是字符串，成交量和持仓量的数据类型是整数。

```
class VtBarData(VtBaseData):
    """K 线数据"""

    #-----
    def __init__(self):
        """Constructor"""
        super(VtBarData, self).__init__()

        self.vtSymbol = EMPTY_STRING    # 品种代码.交易所
        self.symbol = EMPTY_STRING       # 品种代码
        self.exchange = EMPTY_STRING     # 交易所

        self.open = EMPTY_FLOAT          # OHLC
        self.high = EMPTY_FLOAT
        self.low = EMPTY_FLOAT
```



```

self.close = EMPTY_FLOAT

self.date = EMPTY_STRING      # 日期
self.time = EMPTY_STRING      # 时间
self.datetime = None          # datetime 时间对象

self.volume = EMPTY_INT       # 成交量
self.openInterest = EMPTY_INT # 持仓量

```

### 3. loadMCCsv 用来把 MultiCharts 下载的历史数据插入 MongoDB 数据库

函数参数有 3 个：fileName、dbName、symbol。

```
def loadCsv(fileName, dbName, symbol):
```

如果这 3 个参数分别是“IF0000\_1min.csv”、MINUTE\_DB\_NAME、IF0000，则该函数的作用就是把沪深 300 股指期货（IF）1 分钟 K 线级别数据的 csv 文件导入 MongoDB 数据库中的 1 分钟 K 线文件夹（MINUTE\_DB\_NAME）里，并生成名为 IF0000 的文档对象。

（1）在主函数中，记录插入数据的总耗时。

记录开始的时间 start = time()，插入数据完成后再计时一次 time()。这样就可以通过 time() - start 计算出导入数据的总耗时是多少了。

```

start = time()
print u'开始读取 CSV 文件%s 中的数据插入到%s 的%s 中' %(fileName, dbName, symbol)
.....
print u'插入完毕, 耗时: %s' % (time()-start)

```

（2）在主函数中，锁定集合，并创建索引。

传统的关系数据库一般由数据库（database）、表（table）、记录（record）组成，而 MongoDB 是由数据库（database）、集合（collection）、文档（document）组成的。MongoDB 用集合来替代关系数据库里的表，但是集合中没有列、行和关系的概念，这体现了模型自由的特点。

同时，MongoDB 以 bson 格式存储文档（Documents）。它很像 JavaScript 中

定义的 json 格式，不过在数据存储的时候，MongoDB 数据库为文档增加了序列化的操作。

以下代码将 MongoClient 设置为本地数据库，集合对象是[dbName]中的[symbol]，ensure\_index()是保证其索引的唯一性。

```
# 锁定集合，并创建索引
client = pymongo.MongoClient(globalSetting['mongoHost'],
globalSetting['mongoPort'])
collection = client[dbName][symbol]
collection.ensure_index([('datetime', pymongo.ASCENDING)], unique=True)
```

(3) 在主函数中，读取数据并将其插入数据库。

用 DictReader()方法读取数据，返回的是一个字典。如果用 reader()方法，则返回的就是一个迭代对象。

使用 K 线数据模板 (VtBarData) 遍历数据，其顺序为 VtSymbol (symbol) → K 线高开低收数据 → datetime → 成交量。

插入一个过滤器 flt，update\_one 保证其唯一性（所谓数据库的唯一性，就是在这个键已知时所对应的数值是唯一的，若重复插入，则要么报错，要么新的值覆盖老的值），{'\$set':bar.\_\_dict\_\_} 表示 flt 出现的时候做更新，upsert=True 意即当数据已经存在的时候去覆盖它；upsert=False（默认）为当数据已经存在的时候，不做任何操作。

```
# 读取数据和插入数据库
reader = csv.DictReader(file(fileName, 'r'))
for d in reader:
    bar = VtBarData()
    bar.vtSymbol = symbol
    bar.symbol = symbol
    bar.open = float(d['Open'])
    bar.high = float(d['High'])
    bar.low = float(d['Low'])
    bar.close = float(d['Close'])
    bar.date = datetime.strptime(d['Date'], '%Y-%m-%d').strftime('%Y%m%d')
    bar.time = d['Time']
```



```

bar.datetime = datetime.strptime(bar.date + ' ' + bar.time,
'%Y%m%d %H:%M:%S')
bar.volume = d['TotalVolume']

flt = {'datetime': bar.datetime}
collection.update_one(flt, {'$set':bar.__dict__}, upsert=True)

```

图 4-1 展示的是在 Jupyter Notebook 上的运行效果：把 2 个 CSV 文件导入 MongoDB 数据库，分别是沪深 300 股指期货和螺纹钢期货的 1 分钟 K 线级别数据。导入 MongoDB 的时间会比较长，使用固态硬盘可以提高速度。

```

loadCsv('IF0000_1min.csv', MINUTE_DB_NAME, 'IF0000')
loadCsv('rb0000_1min', MINUTE_DB_NAME, 'rb0000')
开始读取CSV文件IF0000_1min.csv中的数据插入到VnTrader_1Min_Db的IF0000中
20100416 09:16:00
20100416 09:17:00
20100416 09:18:00
20100416 09:19:00
20100416 09:20:00
20100416 09:21:00
20100416 09:22:00
20100416 09:23:00

```

图 4-1 把 CSV 文件导入 MongoDB 数据库

## 4.1.2 开发新的 CSV 导入模块

基于 vn.py 官方提供的 CSV 数据加载模块，用户可以自行开发出新的数据导入模块，下面以 Mc8s 版为例具体说明。

从 Mc8s 版中下载 CSV 格式的沪铜 1 分钟 K 线数据，用 Excel 打开，如图 4-2 所示。

	A	B	C	D	E	F	G	H
1	<Date>	<Time>	<Open>	<High>	<Low>	<Close>	<Volume>	
2	2014/7/14	9:01:00	1104	1109	1104	1107	26330	
3	2014/7/14	9:02:00	1106	1108	1105	1108	10388	
4	2014/7/14	9:03:00	1107	1108	1106	1107	7040	
5	2014/7/14	9:04:00	1107	1108	1105	1108	9176	
6	2014/7/14	9:05:00	1108	1108	1106	1107	7908	
7	2014/7/14	9:06:00	1107	1110	1106	1109	7126	
8	2014/7/14	9:07:00	1110	1111	1108	1109	6700	
9	2014/7/14	9:08:00	1109	1111	1109	1109	6846	
10	2014/7/14	9:09:00	1109	1110	1107	1108	7238	
11	2014/7/14	9:10:00	1108	1108	1106	1107	3070	
12	2014/7/14	9:11:00	1108	1108	1107	1108	4638	
13	2014/7/14	9:12:00	1108	1108	1107	1107	144	
14	2014/7/14	9:13:00	1107	1108	1107	1107	542	
15	2014/7/14	9:14:00	1107	1108	1105	1106	6394	
16	2014/7/14	9:15:00	1106	1107	1105	1107	4944	

图 4-2 沪铜 1 分钟 K 线数据

发现数据格式基本没问题，但是为了方便导入 MongoDB 中，需要修改表头和删除日期列的星期字段。

### 1. 修改表头

用 Sublime Text 打开 CSV 文件，修改表头。用 Sublime Text 编辑的好处是不会修改其数据格式，如图 4-3 所示。

```
1 "Date","Time","Open","High","Low","Close","Volume"
2 2014/7/14 09:01:00,1104.0,1109.0,1104.0,1107.0,26330
3 2014/7/14 09:02:00,1106.0,1108.0,1105.0,1108.0,19388
4 2014/7/14 09:03:00,1107.0,1108.0,1106.0,1107.0,7040
5 2014/7/14 09:04:00,1107.0,1108.0,1105.0,1108.0,9176
6 2014/7/14 09:05:00,1108.0,1108.0,1106.0,1107.0,7908
7 2014/7/14 09:06:00,1107.0,1110.0,1106.0,1109.0,7126
8 2014/7/14 09:07:00,1110.0,1111.0,1108.0,1109.0,6700
9 2014/7/14 09:08:00,1109.0,1111.0,1109.0,1109.0,6846
10 2014/7/14 09:09:00,1109.0,1110.0,1107.0,1108.0,7238
11 2014/7/14 09:10:00,1108.0,1108.0,1106.0,1107.0,3070
12 2014/7/14 09:11:00,1108.0,1108.0,1107.0,1108.0,4638
13 2014/7/14 09:12:00,1108.0,1108.0,1107.0,1107.0,144
14 2014/7/14 09:13:00,1107.0,1108.0,1107.0,1107.0,542
15 2014/7/14 09:14:00,1107.0,1108.0,1105.0,1106.0,6394
16 2014/7/14 09:15:00,1106.0,1107.0,1105.0,1107.0,4944
17 2014/7/14 09:16:00,1107.0,1109.0,1107.0,1108.0,4564
18 2014/7/14 09:17:00,1108.0,1108.0,1107.0,1107.0,4820
19 2014/7/14 09:18:00,1108.0,1108.0,1107.0,1107.0,688
20 2014/7/14 09:19:00,1107.0,1109.0,1106.0,1107.0,5630
21 2014/7/14 09:20:00,1106.0,1107.0,1106.0,1107.0,1876
22 2014/7/14 09:21:00,1107.0,1107.0,1106.0,1107.0,290
23 2014/7/14 09:22:00,1107.0,1107.0,1107.0,1107.0,458
```

图 4-3 用 Sublime Text 打开 CSV 文件

### 2. 导入 MongoDB 数据库

#### (1) 导入库。

```
import csv
from datetime import datetime, timedelta
from time import time

import pymongo

from vnpy.trader.vtGlobal import globalSetting
from vnpy.trader.vtConstant import *
from vnpy.trader.vtObject import VtBarData
from vnpy.trader.app.ctaStrategy.ctaBase import SETTING_DB_NAME, TICK_DB_NAME,
MINUTE_DB_NAME, DAILY_DB_NAME
```



## (2) 构造函数。

```
def loadMc8sCsv(fileName, dbName, symbol):
    """将 MC8s 版本导出的 txt 格式的历史数据插入 MongoDB 数据库中"""
    import csv

    start = time()
    print u'开始读取 CSV 文件%s 中的数据插入到%s 的%s 中' %(fileName, dbName, symbol)

    # 锁定集合并创建索引
    client = pymongo.MongoClient(globalSetting['mongoHost'],
    globalSetting['mongoPort'])
    collection = client[dbName][symbol]
    collection.ensure_index([('datetime', pymongo.ASCENDING)], unique=True)

    # 读取数据和插入数据库
    reader = csv.DictReader(file(fileName, 'r'))
    for d in reader:
        bar = VtBarData()
        bar.vtSymbol = symbol
        bar.symbol = symbol

        bar.open = float(d['Open'])
        bar.high = float(d['High'])
        bar.low = float(d['Low'])
        bar.close = float(d['Close'])
        bar.date = datetime.strptime(d['Date'].split(' ')[0],
        '%Y/%m/%d').strftime('%Y%m%d')

        bar.time = d['Time']
        bar.datetime = datetime.strptime(bar.date + ' ' + bar.time,
        '%Y%m%d %H:%M:%S')
        bar.volume = d['Volume']

        flt = {'datetime': bar.datetime}
        collection.update_one(flt, {'$set': bar.__dict__}, upsert=True)
        print bar.date, bar.time

    print u'插入完毕, 耗时: %s' % (time()-start)
```

## (3) 在 Jupyter Notebook 中输入下面的代码即可开始导入。

```
loadMc8sCsv('j0000.csv', MINUTE_DB_NAME, 'j0000')
```





### 4.1.3 数据下载模块

vn.py 提供 5 种数据下载模块，分别对应不同的品种。

- 天勤（TqDataService）：期货。
- 上海中期（ShcifcoDataService）：期货。
- 富途（FutuDataService）：港股、美股。
- Tushare（TushareDataService）：国内股票。
- CoinAPI（CoinapiDataService）：数字货币。

它们处于“vnpy-1.9.0\examples”文件夹下。数据下载又分为手动下载和自动定时下载。下面以天勤终端数据下载模块为例，具体讲解手动下载和自动定时下载的功能。

#### 1. 安装天勤终端

天勤数据下载模块需要终端运行后才能使用，可去天勤官网下载终端安装包，安装完成后会出现如图 4-4 所示的天勤终端界面。



图 4-4 天勤终端界面

#### 2. 手动下载模块

手动下载模块在 downloadData.py 文件中，只需要填写合约代码和 1 分钟 K



线的数量即可。

```
from dataService import *
if __name__ == '__main__':
    symbols = ["SHFE.cu1901"]
    downloadAllMinuteBar(1000, symbols)
```

运行时会调用在同一文件夹 dataService.py 的 downloadAllMinuteBar 函数进行下载，天勤终端下载成功界面如图 4-5 所示。

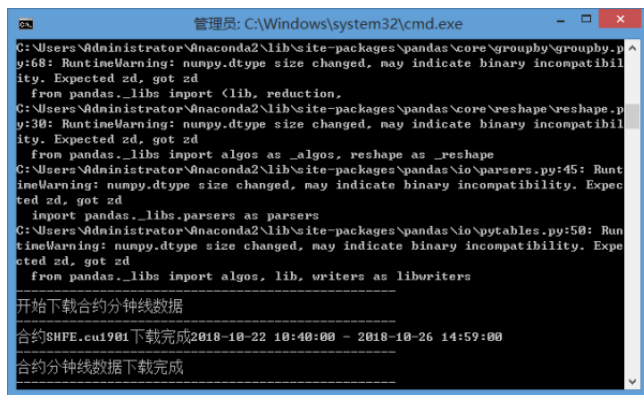


图 4-5 天勤终端下载成功界面

### 3. 自动定时下载模块

自动定时下载模块在 runService.py 文件中，同理只要输入合约代码，如 CFFEX.IF1810，启动程序后定时在每个交易日 17 点的某一分钟（随机的任务下载时间）自动下载更新历史行情数据，1000 条 1 分钟 K 线数据足以覆盖两天的行情。

```
if __name__ == '__main__':
    taskCompletedDate = None

    # 生成一个随机的任务下载时间，用于避免所有用户在同一时间访问数据服务器
    taskTime = datetime.time(hour=17, minute=0)

    symbols = ["CFFEX.IF1810", "CFFEX.IF1811", "CFFEX.IF1812",
               "CFFEX.IH1810", "CFFEX.IH1811", "CFFEX.IH1812",
               "CFFEX.IC1810", "CFFEX.IC1811", "CFFEX.IC1812",]
    # 进入主循环
```

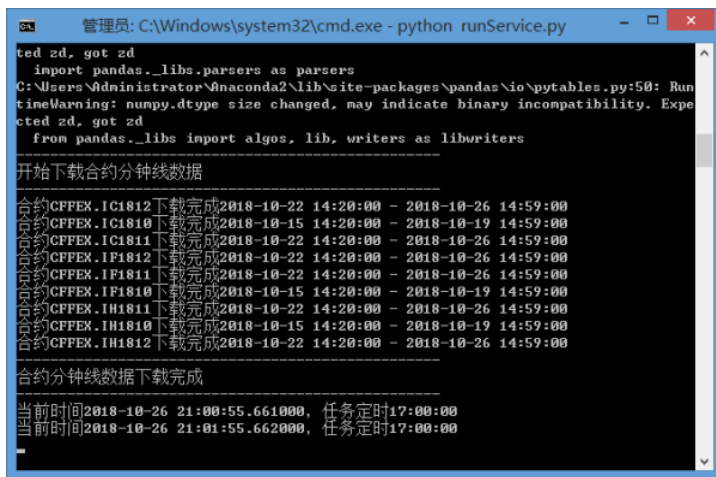
```
while True:
    t = datetime.datetime.now()

    # 每天到达任务下载时间后, 执行数据下载的操作
    if t.time() > taskTime and (taskCompletedDate is None or t.date() !=
taskCompletedDate):
        # 下载 1000 条 1 分钟 K 线数据, 足以覆盖两天的行情
        downloadAllMinuteBar(1000, symbols)

        # 更新任务完成的日期
        taskCompletedDate = t.date()
    else:
        print(u'当前时间%s, 任务定时%s' %(t, taskTime))

    time.sleep(60)
```

运行程序后实现的天勤终端自动定时下载如图 4-6 所示。



```
管理员: C:\Windows\system32\cmd.exe - python runService.py
ted zd, got zd
import pandas._libs.parsers as parsers
C:\Users\Administrator\Anaconda2\lib\site-packages\pandas\io\pytables.py:50: Run
timeWarning: numpy.dtype size changed, may indicate binary incompatibility. Expe
cted zd, got zd
from pandas._libs import algos, lib, writers as libwriters

开始下载合约分钟线数据
合约CFFEX.IC1812下载完成2018-10-22 14:20:00 - 2018-10-26 14:59:00
合约CFFEX.IC1810下载完成2018-10-15 14:20:00 - 2018-10-19 14:59:00
合约CFFEX.IC1811下载完成2018-10-22 14:20:00 - 2018-10-26 14:59:00
合约CFFEX.IF1812下载完成2018-10-22 14:20:00 - 2018-10-26 14:59:00
合约CFFEX.IF1811下载完成2018-10-22 14:20:00 - 2018-10-26 14:59:00
合约CFFEX.IF1810下载完成2018-10-15 14:20:00 - 2018-10-19 14:59:00
合约CFFEX.IH1811下载完成2018-10-22 14:20:00 - 2018-10-26 14:59:00
合约CFFEX.IH1810下载完成2018-10-15 14:20:00 - 2018-10-19 14:59:00
合约CFFEX.IH1812下载完成2018-10-22 14:20:00 - 2018-10-26 14:59:00

合约分钟线数据下载完成
当前时间2018-10-26 21:00:55.661000, 任务定时17:00:00
当前时间2018-10-26 21:01:55.662000, 任务定时17:00:00
```

图 4-6 天勤终端自动定时下载

## 4.2 K 线生成模块

在期货市场中, 交易系统会实时收到交易所每秒 2 次的 Tick 行情推送, 其行情信息有开盘价、最高价、最低价、最新价、成交量、成交额、买一价、买一量、



卖一价、卖一量。一般的 CTA 策略都是基于分钟 K 线级别数据做判断的。所以就产生了利用 Tick 级别数据来合成 1 分钟 K 线级别数据的需求。

在 1 分钟 K 线的基础上，也可以合成更加常用的 5 分钟、15 分钟、30 分钟、60 分钟、4 小时、1 日 K 线，只有包含充足的各种周期的 K 线序列，CTA 策略才能够正常运作起来，从而产生交易信号。

同样，在策略回测中，也能根据存放在 MongoDB 数据库中的 Tick 级别数据或者 1 分钟 K 线级别数据来合成各个周期的 K 线数据来进行回测，然后基于回测结果，对 K 线周期进行优化。例如某些品种，用 13 分钟 K 线“跑”出的结果远远好于用 15 分钟 K 线“跑”出的结果。

K 线生成模块处于“vnpy-1.9.0\vnpy\trader\app\ctaStrategy”文件夹的 ctaTemplate.py 文件内，该模块分成 2 部分，分别是 1 分钟 K 线合成及 X 分钟 K 线合成。1.9.2 版本中，K 线生成模块和 K 线时间序列管理模块则从 ctaTemplate 中独立开来，放在“vnpy-1.9.2-LTS\vnpy\trader”文件夹内的 vtUtility.py 中。

## 4.2.1 1 分钟 K 线合成

定义 Tick 级别数据更新函数（updateTick）的作用是当 Tick 数据推送到 1 分钟的时候会自动合成新的 1 分钟 K 线。

根据判断标准 `self.bar.datetime.minute != tick.datetime.minute` 来进行时间切片：若当前 1 分钟还没走完，则调用 K 线数据模板（VtBarData），继续收录行情推送，生成上 1 分钟 K 线。若开始新的 1 分钟，则初始化新的 K 线对象。

生成上 1 分钟 K 线的流程如下。

（1）用方法 `datetime.replace()` 把秒和微秒设为 0，方便处理分钟级别 K 线的数据。

（2）用 `datetime.strftime()` 方法把数据格式化为字符串显示给用户，其中：

- date 类型为 `%Y%m%d`（%Y 表示带有世纪的年份，%m 表示月份数，以 0 填

充的十进制数，%d 表示这个月的第几天，以 0 填充的十进制数），比如 20180717；

- time 类型为%H:%M:%S.%f( %H 表示 24 小时制的小时数，%M 表示分钟，%S 表示秒，%f 表示微秒)，因为上一步已经把秒和微秒设为 0，所以这里可以表示为 10:01:00.000000。

(3) 把刚刚走完的 1 分钟 K 线对象推送到 onBar 函数。

(4) 创建最新的 1 分钟 K 线对象。

```
# 尚未创建对象
if not self.bar:
    self.bar = VtBarData()
    newMinute = True
# 新的 1 分钟
elif self.bar.datetime.minute != tick.datetime.minute:
    # 生成上 1 分钟 K 线的时间戳
    self.bar.datetime = self.bar.datetime.replace(second=0, microsecond=0)
    # 将秒和微秒设为 0
    self.bar.date = self.bar.datetime.strftime('%Y%m%d')
    self.bar.time = self.bar.datetime.strftime('%H:%M:%S.%f')

    # 推送已经结束的上 1 分钟 K 线
    self.onBar(self.bar)

    # 创建新的 K 线对象
    self.bar = VtBarData()
    newMinute = True
```

当判断为新的 1 分钟后，初始化新的 K 线对象，即把 Tick 级别数据中的期货合约和交易所的信息分别存入 vtSymbol、symbol 和 exchange 中（不像股票代码，期货合约代码具有唯一性，所以 vtSymbol 等于 symbol），然后开始缓存第一条 Tick 数据的开盘价、最高价和收盘价（最后才处理收盘价和成交量）。

若 newMinute = False，则表示 1 分钟 K 线正在生成，还没有走完，初始化完开盘价、最高价和最低价之后要统计出最高价和收盘价。



最高价  $\max(\text{self.bar.high}, \text{tick.lastPrice})$  表示缓存进来的  $\text{bar.high}$  与每一次新进来的 Tick 对比, 取最大值。同理, 最低价  $\min(\text{self.bar.low}, \text{tick.lastPrice})$  表示缓存进来的  $\text{bar.low}$  与每一次新进来的 Tick 对比, 取最小值。

```
# 初始化新 1 分钟的 K 线数据
if newMinute:
    self.bar.vtSymbol = tick.vtSymbol
    self.bar.symbol = tick.symbol
    self.bar.exchange = tick.exchange

    self.bar.open = tick.lastPrice
    self.bar.high = tick.lastPrice
    self.bar.low = tick.lastPrice
# 累加更新老 1 分钟的 K 线数据
else:
    self.bar.high = max(self.bar.high, tick.lastPrice)
    self.bar.low = min(self.bar.low, tick.lastPrice)
```

最后步骤是通用部分的更新, 分别缓存收盘价、日期时间(如 20180717 10:01:00.000000) 和未平仓合约。

$\text{self.bar.volume} += \max(\text{volumeChange}, 0)$  的作用是每一次 Tick 推送的时候把成交的变化量累加过来,  $\max(\text{volumeChange}, 0)$  的作用是把负的成交量过滤掉。出现负的成交量的原因是夜盘开盘  $\text{lastTick.volume}$  为昨日收盘数据。

```
# 通用更新部分
self.bar.close = tick.lastPrice
self.bar.datetime = tick.datetime
self.bar.openInterest = tick.openInterest

if self.lastTick:
    volumeChange = tick.volume - self.lastTick.volume # 当前 K 线内的成交量
    self.bar.volume += max(volumeChange, 0)
# 避免夜盘开盘 lastTick.volume 为昨日收盘数据, 导致成交量变化为负的情况

# 缓存 Tick
self.lastTick = tick
```

## 4.2.2 X分钟 K 线合成

定义 1 分钟 K 线的更新函数( updateBar )根据 1 分钟 K 线数据合成 X 分钟 K 线。若还没创建 X 分钟 K 线对象,那就调用 VtBarData 进行创建,然后初始化这些数据:合约名称、交易所代码、K 线的开盘价、最高价和最低价,以及日期时间。若 X 分钟 K 线已创建,则更新最高价和最低价。

要更新的通用部分包括收盘价、未平仓合约和累计成交量。

通过求余(%)的方法对时间进行切片。若合成 5 分钟 K 线,则代表 0~4 分钟,因为(4+1)除以 5 的余数为 0,即 False,表示 5 分钟 K 线刚刚走完了。在 5 分钟 K 线生成后,用 datetime.replace()的方法把秒和微秒设为 0,同时用 datetime.strftime()方法把数据格式化为字符串,比如显示为 20180717 10:01:00.000000。推送 X 分钟 K 线到 onBar 函数,然后清空缓存。

```
def updateBar(self, bar):
    """1 分钟 K 线更新"""
    # 尚未创建对象
    if not self.xminBar:
        self.xminBar = VtBarData()

        self.xminBar.vtSymbol = bar.vtSymbol
        self.xminBar.symbol = bar.symbol
        self.xminBar.exchange = bar.exchange

        self.xminBar.open = bar.open
        self.xminBar.high = bar.high
        self.xminBar.low = bar.low

        self.xminBar.datetime = bar.datetime    # 以第一条 1 分钟 K 线的开始时间戳作为
x 分钟 K 线的时间戳
        # 累加老 K 线
    else:
        self.xminBar.high = max(self.xminBar.high, bar.high)
        self.xminBar.low = min(self.xminBar.low, bar.low)

    # 通用部分
    self.xminBar.close = bar.close
```



```

self.xminBar.openInterest = bar.openInterest
self.xminBar.volume += int(bar.volume)

# x 分钟已经走完
if not (bar.datetime.minute + 1) % self.xmin: # 可以用 x 整除
    # 生成上一 x 分钟 K 线的时间戳
    self.xminBar.datetime = self.xminBar.datetime.replace(second=0,
microsecond=0) # 将秒和微秒设为 0
    self.xminBar.date = self.xminBar.datetime.strftime('%Y%m%d')
    self.xminBar.time = self.xminBar.datetime.strftime('%H:%M:%S.%f')

    # 推送
    self.onXminBar(self.xminBar)

    # 清空老 K 线缓存对象
    self.xminBar = None

```

## 4.3 K 线管理模块

本节主要介绍如何将 K 线数据缓存到 NumPy 的一维数组 Array，形成时间序列，通过调用金融库 TA-Lib 来计算常用技术指标。

K 线管理模块同样在 ctaTemplate.py 文件中，定义为 ArrayManager 类。

### 4.3.1 初始化参数

size = 100 的意思是默认缓存 K 线数量为 100 条。初始化状态为 False（False 表示缓存计数小于 100）。创建 NumPy 的 Array，其开盘价、最高价、最低价、收盘价和成交量都是由一系列（默认为 100）为 0 的数组构成的。

```

def __init__(self, size=100):
    """Constructor"""
    self.count = 0 # 缓存计数
    self.size = size # 缓存大小
    self.inited = False # True if count>=size

    self.openArray = np.zeros(size) # OHLC

```



```
self.highArray = np.zeros(size)
self.lowArray = np.zeros(size)
self.closeArray = np.zeros(size)
self.volumeArray = np.zeros(size)
```

### 4.3.2 生成时间序列

定义 K 线更新函数（updateBar）用于缓存 K 线数据到 Array 数组。

在第 1 条 K 线缓存下来后，进行逻辑判断“if not self.inited and self.count >= self.size”，其格式为“非 a 为真 且 b 为真”，在这里，a = self.inited 为 False，非 a 为 True，所以逻辑判断简化为当 count >= 100 为真时，初始化状态变成 True。

具体流程如下：

第 1 条 K 线 count = count(0) + 1 = 1, 1 >= 100 为 False，在 100 列所有为 0 的数组全部前移 1 位，插入最新的 open、high、close 和 volume（此时有 99 列为 0 的数组）。

第 2 条 K 线 count = count(1) + 1 = 2, 2 >= 100 为 False，在 100 列数组全部前移 1 位，插入最新的 open，high，close 和 volume（此时有 98 列为 0 的数组）。

... ..

第 100 条 K 线 count = count(99) + 1 = 100, 100 >= 100 为 True，立刻更新初始化为 True，通知交易系统界面初始化成功。同时，在 100 列数组全部前移 1 位，插入最新的 open、high、close 和 volume（此时 100 列数组全部缓存了 K 线数据）。

```
def updateBar(self, bar):
    """更新K线"""
    self.count += 1
    if not self.inited and self.count >= self.size:
        self.inited = True

    self.openArray[0:self.size-1] = self.openArray[1:self.size]
    self.highArray[0:self.size-1] = self.highArray[1:self.size]
    self.lowArray[0:self.size-1] = self.lowArray[1:self.size]
```



```

self.closeArray[0:self.size-1] = self.closeArray[1:self.size]
self.volumeArray[0:self.size-1] = self.volumeArray[1:self.size]

self.openArray[-1] = bar.open
self.highArray[-1] = bar.high
self.lowArray[-1] = bar.low
self.closeArray[-1] = bar.close
self.volumeArray[-1] = bar.volume

```

### 4.3.3 定义属性函数

“@property” 可以将 Python 定义的函数当属性来访问，从而提供更加友好的访问方式。将包含 100 条 K 线数据的开盘价、最高价、最低价、收盘价和成交量 的方法变成类的属性，以更加容易被调用。

```

-----
@property
def open(self):
    """获取开盘价序列"""
    return self.openArray

#-----
@property
def high(self):
    """获取最高价序列"""
    return self.highArray

#-----
@property
def low(self):
    """获取最低价序列"""
    return self.lowArray

#-----
@property
def close(self):
    """获取收盘价序列"""
    return self.closeArray

#-----

```

```
@property
def volume(self):
    """获取成交量序列"""
    return self.volumeArray
```

### 4.3.4 生成计算指标

TA-Lib 库的全称是 Technical Analysis Library，主要功能是计算股价的技术分析指标。作为一套被业界广泛应用的开源技术分析库（包含技术指标计算和 K 线模式识别等），TA-Lib 库中一共包含大约 125 个技术指标的计算函数，同时提供了包括 C/C++、Java、Perl、Python 等多种语言的 API。

该金融库使用起来非常方便，只要输入参数就能得到技术指标的数值了。为了方便演示，需要打开 Wing IDE 的 Python Shell 界面。

#### 例子 1：均线计算

简单移动均线（Simple Moving Average，SMA），又称算术移动均线，是指对特定期间的收盘价进行简单平均化。一般提及的移动均线即指简单移动均线（SMA）。简单移动均线沿用最简单的统计学方式，将过去某特定时间内的价格取其平均值。简单移动均线的计算方法如同其名一样简单，它只是将每日得到的平均值连成一线并随时间移动，每一支蜡烛线因而得到相同的数值。

在 Python Shell 中输入下面命令，然后按 Enter 键会出现如图 4-7 所示的界面，展示了 TA-Lib 中对 SMA 函数的定义。

```
import talib
help(talib.SMA)
```

在 TA-Lib 中，均线函数 SMA 入参 2 个：收盘价和计算均线的周期。默认周期是 30，出参是计算出来的均值。



```
>>> import talib
>>> help(talib.SMA)
Help on built-in function SMA in module talib.func:

SMA(...)
    SMA(real[, timeperiod=?])

    Simple Moving Average (Overlap Studies)

    Inputs:
        real: (any ndarray)
    Parameters:
        timeperiod: 30
    Outputs:
        real
>>> |
```

图 4-7 SMA 函数的定义

定义函数 sma（注意，这里函数名称是小写），入参是周期数 n 和 array（默认为 False）。调用 TA-Lib 的 SMA 函数计算均线。若 array 为默认值（即 False），则只计算最新一条均线的数值，否则计算出所有均线的数值。

```
def sma(self, n, array=False):
    """简单均线"""
    result = talib.SMA(self.close, n)
    if array:
        return result
    return result[-1]
```

### 例子 2: RSI 计算

RSI 指标（Relative Strength Index）是由韦尔斯·怀尔德（Welles Wilder）提出的，是衡量证券自身内在相对强度的指标。它是根据一定时期内上涨和下跌幅度之和的比率制作出的技术曲线，能够反映出市场在一定时期内的景气程度。

同理，在 Python Shell 中输入下面命令，然后按 Enter 回车键会出现如图 4-8 所示的界面，展示了 TA-Lib 中对 RSI 函数的定义。

```
help(talib.RSI)
```

```
>>> help(talib.RSI)
Help on built-in function RSI in module talib.func:

RSI(...)
    RSI(real[, timeperiod=?])

    Relative Strength Index (Momentum Indicators)

    Inputs:
        real: (any ndarray)
    Parameters:
        timeperiod: 14
    Outputs:
        real

>>> |
```

图 4-8 RSI 函数定义

在 TA-Lib 中，RSI 函数入参 2 个：收盘价和统计周期，默认周期是 14，输出是 RSI 值。定义函数 rsi 入参是周期数 n 和 array（默认值为 False）。调用 TA-Lib 的 RSI 函数计算均线。若 array 为默认值（即 False），则只计算最新的 RSI 值，否则输出所有 RSI 值。

```
def rsi(self, n, array=False):
    """RSI 指标"""
    result = talib.RSI(self.close, n)
    if array:
        return result
    return result[-1]
```

vn.py 官方已实现的 K 线管理模块常用技术指标包括简单均线、标准差、CCI 指标、ATR 指标、RSI 指标、MACD 指标、ADX 指标、布林通道、肯特那通道和唐奇安通道。其他技术指标需要用户自行添加，添加完毕后需要在 Anaconda 中更新 vn.py 的真实运行目录。

## 4.4 CTA 策略模块

CTA 策略模块是具体 CTA 策略的基础。具体策略（子类）是继承于 CTA 策略模块（父类）进行开发的，增加了个性化设计，如入场价格和止赢止损。CTA 策略模块与前面介绍的 K 线生成模块和 K 线管理模块一样，都在“vnpy-1.9.0\vnpy\



trader\app\ctaStrategy” 文件夹的 ctaTemplate.py 文件内。CTA 策略模块分为 4 部分，分别是定义成员变量、构键函数、回调函数和主动函数。

### 4.4.1 定义成员变量

分别定义如下成员变量：

- 策略类的名称和作者。
- MongoDB 数据库名称。
- 策略的基本参数。
- 参数名称列表。
- 变量名称列表。
- 同步列表。

需要注意的一个问题是，用户定义的参数和变量列表属于类的属性，在实例初始化的时候会直接指向该类的同名变量，导致多个实例共享同一个数据容器，策略执行出错。解决方法是将所有的可变变量的定义（如列表、字典等）放在初始化函数\_\_init\_\_中。

#### 例子 1：错误用法

```
class TestStrategy(CtaTemplate):
    """CTA 策略模板"""
    ...

    # 这里定义的变量是类成员，方便引擎在创建策略对象前了解一些信息

    barList = []    # 若基于 TestStrategy 类创建多个策略对象，它们的 barList 都会指向同一个
                    # 列表，导致出错
    lastPrice = 0    # 数字(int、float)在 Python 中属于不可变对象，因此每个策略的 lastPrice
                    # 互不影响

    #-----
    def __init__(self, ctaEngine, setting):
        super(TestStrategy, self).__init__(ctaEngine, setting)
```

### 例子 2: 正确用法

```
class TestStrategy(CtaTemplate):
    """CTA 策略模板"""
    ...

    # 这里定义的变量是类成员, 方便引擎在创建策略对象前了解一些信息

    barList = []    # 若基于 TestStrategy 类创建多个策略对象, 它们的 barList 都会指向同一个
                    # 列表, 导致出错
    lastPrice = 0    # 数字(int、float)在 Python 中属于不可变对象, 因此每个策略的 lastPrice
                    # 互不影响

    #-----
    def __init__(self, ctaEngine, setting):
        super(TestStrategy, self).__init__(ctaEngine, setting)

    # 这里定义的变量是对象创建后自身独有的成员

    self.barList = []    # 在这里对 barList 重新初始化, 指向一个新建的独立列表
```

另一个问题是无法多品种运用同一个 CTA 策略。解决方案有三, 分别是在 VnTrader 中设置 CTA\_setting.json 实盘配置文件、策略内部实现, 以及多策略实例模式。其中以多策略实例模式最为简单, 就是把原策略文件复制多份, 并且修改文件名和策略类名, 里面的变量参数及策略逻辑无须修改, 复制好的都对应对着一个品种即可, 例如:

实例 1	DemoStrategy_rb	品种 rb1901
实例 2	DemoStrategy_TA	品种 TA901
实例 3	DemoStrategy_RM	品种 RM901

设置 CTA\_setting 如图 4-9 所示, 其详细用法将在第 8 章讲述。

## 4.4.2 构造函数

`__init__()` 是策略对象在创建时会被首先调用的构造函数, 这个函数前后都有两个下划线, 传入 `ctaEngine` 实例和 `setting` 参数配置字典来创建策略的初始数据状态。



```

1  [
2      {
3          "name": "MultiTimeFrame_Rb",
4          "className": "MultiTimeframeStrategy",
5          "vtSymbol": "rb1901",
6          "fixedSize": 10,
7          "rsiWindow": 20,
8          "rsiLength": 14,
9          "fastWindow": 8,
10         "slowWindow": 10
11     },
12
13     {
14         "name": "MultiTimeFrame_Ru",
15         "className": "MultiTimeframeStrategy",
16         "vtSymbol": "ru1901",
17         "fixedSize": 5,
18         "rsiWindow": 24,
19         "rsiLength": 14,
20         "fastWindow": 12,
21         "slowWindow": 20
22     }
23 ]
24

```

图 4-9 设置 CTA\_setting

`__init__()`的作用是在创建类的实例的时候，实例会自动调用这个方法，一般用来对实例的属性进行初始化。`__init__()`方法的第一个参数永远都是 `self`，表示创建实例本身，在`__init__()`方法内部，可以把各种属性绑定到 `self`，因为 `self` 指向创建的实例本身。`super` 用于继承父类，其写法是 `super(子类, self).__init__(参数 1, 参数 2, ...)`

```

def __init__(self, ctaEngine, setting):
    """Constructor"""
    self.ctaEngine = ctaEngine

    # 设置策略的参数
    if setting:
        d = self.__dict__
        for key in self.paramList:
            if key in setting:
                d[key] = setting[key]

```

### 4.4.3 回调函数

回调函数如下所述。

- `onInit`: 在创建实例后，如有需要还可以用该方法进一步初始化，例如加载历史数据计算策略变量状态。



- `onStart`: 策略启动方法, 一般做一些启动提示。
- `onStop`: 策略停止方法, 如撤单、平今仓转平昨仓、结算收盘等。
- `onTick(self, tick)`: 当最新 tick 数据更新时, 做一些计算, 信号触发, 当然也可以做撤单。收盘后的一段时间可能收到垃圾数据 (因为有时柜台系统会做一些调试), 建议退出账户, 也可在策略或者引擎里自行处理。以 tick 为实例, 获取属性请使用对象方法, 如 “`tick.lastPrice`”, 下面的 `order`、`trade`、`bar` 皆如此。
- `onOrder(self, order)`: 当发出委托后, 就会收到关于委托状态的信息, 无论是否成交, 可以根据回报的信息来进行撤单、追单等处理。
- `onTrade(self, trade)`: 成交数据回报。主要用来计算持仓, 也可以用来触发委托、追单等操作。
- `onBar(self, bar)`: 该方法一般在 `onTick` 里被调用, 当生成新 K 线的时候, 触发该方法, 进而触发基于 K 线的策略信号。

#### 4.4.4 主动函数

主动函数如下所述。

- `buy(self, price, volume, stop=False)`: 开仓做多, 默认合约为 `self.vtSymbol`, `stop` 为 `False` 直接发市价单, 为 `True` 时发本地停止单, 默认 `False`。
- `sell(self, price, volume, stop=False)`: 平多仓, 默认市价单平仓。
- `short(self, price, volume, stop=False)`: 开仓做空, 默认市价单开仓。
- `cover(self, price, volume, stop=False)`: 平空仓, 默认市价单平仓。
- `sendOrder(self, orderType, price, volume, stop=False)`: 区别对待本地停止单和直接发市价单。所以该函数是前面 `buy`, `sell`, `short`, `over` 函数的基础, 二次封装后被其调用。
- `cancelOrder(self, vtOrderID)`: 根据发单编号进行撤单, 该方法根据 `vtOrderID` 来判断撤本地单子, 还是撤交易所排队的单子。
- `insertTick(self, tick)`: 向数据库中插入 tick 数据。对 `ctaEngine` 函数的二次封装,



方便使用，以下 3 个也是如此。

- `insertBar(self, bar)`: 向数据库中插入 bar 数据。
- `loadTick(self, days)`: 根据天数读取 tick 数据。
- `loadBar(self, days)`: 载入 bar 数据用于策略初始化，一般 CTA 策略默认载入 10 天数据。
- `writeCtaLog(self, content)`: 对 `ctaEngine` 日志方法进行二次封装，主要加上策略的名称，用于区分多策略日志。
- `putEvent`: 发出策略状态变化信号，主要通知监控系统。
- `getEngineType`: 区分引擎类型，满足不同情况的处理。

```
#-----
def buy(self, price, volume, stop=False):
    """买开"""
    return self.sendOrder(CTAORDER_BUY, price, volume, stop)

#-----
def sell(self, price, volume, stop=False):
    """卖平"""
    return self.sendOrder(CTAORDER_SELL, price, volume, stop)

#-----
def short(self, price, volume, stop=False):
    """卖开"""
    return self.sendOrder(CTAORDER_SHORT, price, volume, stop)

#-----
def cover(self, price, volume, stop=False):
    """买平"""
    return self.sendOrder(CTAORDER_COVER, price, volume, stop)

#-----
def sendOrder(self, orderType, price, volume, stop=False):
    """发送委托"""
    if self.trading:
        # 如果 stop 为 True, 则意味着发本地停止单
        if stop:
            vtOrderIDList = self.ctaEngine.sendStopOrder(self.vtSymbol,
orderType, price, volume, self)
```

```

        else:
            vtOrderIDList = self.ctaEngine.sendOrder(self.vtSymbol, orderType,
price, volume, self)
            return vtOrderIDList
        else:
            # 交易停止时发单返回空字符串
            return []

#-----
def cancelOrder(self, vtOrderID):
    """撤单"""
    # 如果发单号为空字符串, 则不进行后续操作
    if not vtOrderID:
        return

    if STOPORDERPREFIX in vtOrderID:
        self.ctaEngine.cancelStopOrder(vtOrderID)
    else:
        self.ctaEngine.cancelOrder(vtOrderID)

#-----
def cancelAll(self):
    """全部撤单"""
    self.ctaEngine.cancelAll(self.name)

#-----
def insertTick(self, tick):
    """向数据库中插入 tick 数据"""
    self.ctaEngine.insertData(self.tickDbName, self.vtSymbol, tick)

#-----
def insertBar(self, bar):
    """向数据库中插入 bar 数据"""
    self.ctaEngine.insertData(self.barDbName, self.vtSymbol, bar)

#-----
def loadTick(self, days):
    """读取 tick 数据"""
    return self.ctaEngine.loadTick(self.tickDbName, self.vtSymbol, days)

#-----
def loadBar(self, days):

```



```

    """读取 bar 数据"""
    return self.ctaEngine.loadBar(self.barDbName, self.vtSymbol, days)

#-----
def writeCtaLog(self, content):
    """记录 CTA 日志"""
    content = self.name + ':' + content
    self.ctaEngine.writeCtaLog(content)

#-----
def putEvent(self):
    """发出策略状态变化事件"""
    self.ctaEngine.putStrategyEvent(self.name)

#-----
def getEngineType(self):
    """查询当前运行的环境"""
    return self.ctaEngine.engineType

```

## 4.5 策略回测模块

交易策略的思路可能源于实盘中的交易经验总结,也可能源于通过数据挖掘、统计分析得到的规律。当验证策略的可行性时,需要用到回测模块。通过回测历史数据,了解策略的历史收益、最大回撤和回撤时长,并且基于回测结果进行参数优化等。

CTA 策略回测模块处于“vnpy-1.9.0\vnpy\trader\app\ctaStrateg”文件夹中的 ctaBacktesting.py 文件内,该模块主要分为两部分,分别是 CTA 回测引擎和参数优化设置。

### 4.5.1 CTA 回测引擎

回测引擎的功能是设置回测参数,初始化策略对象,运行回测并输出结果。函数接口和实盘引擎保持一致,从而实现从回测到实盘同一套策略代码。按照功能,可以将函数分成 4 类,分别是参数设置、数据回放、策略接口和结果计算。

## 1. 参数设置

- `setStartDate(self, startDate='20100416', initDays=10)`: 设置回测开始时间和初始化天数。回测时间默认是 2010 年 4 月 16 日, 初始化天数就是策略回测开始前需要准备多少日的数据, 默认是 10 日。
- `setEndDate(self, endDate='')`: 设置回测结束日期, 默认日期为数据库里的最近日期。
- `setBacktestingMode(self, mode)`: 设置回测模式, 支持两种模式, 一种是 Tick 模式 ( `BacktestingEngine.TICK_MODE` ), 另一种是 K 线模式 ( `BacktestingEngine.BAR_MODE` )。
- `setDatabase(self, dbName, symbol)`: 设置合约数据的数据库名及表名, 以便回测引擎可以载入对应的数据。
- `setCapital(self, capital)`: 设置初始资金。
- `setSlippage(self, slippage)`: 设置每一手合约的滑点, 单位是合约的价格。
- `setSize(self, size)`: 设置合约大小。不同的交易品种对应着不同的合约规模, 例如股指期货是 300 元/点, 螺纹钢期货是 10 吨/手。成交金额=合约价格(面值) $\times$ 合约大小。
- `setRate(self, rate)`: 设置佣金比例, 交易手续费=成交金额 $\times$ 佣金比例。
- `setPriceTick(self, priceTick)`: 设置合约的最小价格变动, 不同的合约对应不同最新价格变动, 如股指期货最小价格变动单位为 0.2 点, 而螺纹钢期货是 1 元/吨。

## 2. 数据回放

- `loadHistoryData(self)`: 载入历史数据。根据回测模式 ( Tick 模式/K 线模式 ), 确认要使用的数据种类, 然后载入初始化需要的数据, 将数据从查询指针中读取出来, 生成列表, 最后载入回测数据。
- `runBacktesting(self)`: 运行回测。从数据库里载入数据, 逐条推入策略做回测, 同时模拟委托和成交, 并保存回测过程的中间数据, 以此可以计算回测结果并显示。
- `newBar(self, bar)`: 生成新的 K 线, 流程是先撮合限价单, 再撮合停止单, 然



后推送 K 线数据到策略中。

- `newTick(self, tick)`: 生成新的 Tick，流程是先撮合限价单，再撮合停止单，然后推送 Tick 数据到策略中。
- `initStrategy(self, strategyClass, setting=None)`: 初始化策略，`setting` 是策略的参数设置，如果使用类中写好的默认设置则可以不传该参数。
- `crossLimitOrder(self)`: 基于最新数据撮合限价单，流程是先确定会撮合成交的价格，遍历限价单字典中的所有限价单，推送委托进入未成交队列的更新状态，判断成交状态，若出现成交，则推送成交数据和委托数据，并且从字典中删除该限价单。
- `crossStopOrder(self)`: 基于最新数据撮合停止单，流程与 `crossLimitOrder` 差不多，应该注意的是确定其撮合成交与限价单撮合的规则相反。

### 3. 策略接口

- `sendOrder(self, vtSymbol, orderType, price, volume, strategy)`: 模拟限价委托单，并把限价单保存在限价单字典和工作限价单字典中。工作限价单字典是需要撮合成交的委托。限价单字典则是所有委托。
- `cancelOrder(self, vtOrderID)`: 模拟撤销限价委托单，并从工作限价单字典中删除。
- `sendStopOrder(self, vtSymbol, orderType, price, volume, strategy)`: 模拟本地停止单，并把停止单保存在停止单字典和工作停止单字典中。工作停止单字典里是需要撮合成交的委托。停止单字典里则是所有委托。
- `cancelStopOrder(self, stopOrderID)`: 模拟撤销停止委托单，并从工作停止单字典中删除。
- `putStrategyEvent(self, name)`: 发送策略更新事件，回测中忽略。
- `insertData(self, dbName, collectionName, data)`: 插入数据到数据库，回测中忽略。
- `loadBar(self, dbName, collectionName, startDate)`: 载入初始化列表中的 K 线级别数据，即策略回测开始前的准备数据。
- `loadTick(self, dbName, collectionName, startDate)`: 载入初始化列表中的 Tick 级别数据，即策略回测开始前的准备数据。
- `writeCtaLog(self, content)`: 记录日志到日志列表。



- `cancelAll(self, name)`: 全部撤单, 包括未成交的限价单和停止单。
- `saveSyncData(self, strategy)`: 保存同步数据。
- `getPriceTick(self, strategy)`: 获取最小价格变动。

#### 4. 结果计算

- `calculateBacktestingResult(self)`: 逐条匹配交易, 计算每笔交易的盈亏。基于每笔交易的结果, 计算具体的盈亏曲线和最大回撤等, 然后再计算盈亏相关数据, 最后返回回测结果。
- `showBacktestingResult(self)`: 显示回测结果, 输出回测报告, 并显示回测结果图, 包括资金子图、回撤子图、每笔交易盈亏子图和每笔盈亏分布子图。
- `clearBacktestingResult(self)`: 清空之前的回测结果, 包括相关的限价单、相关的停止单和相关成交。
- `runOptimization(self, strategyClass, optimizationSetting)`: 运行普通模式的优化参数, 也就是说 Python 主进程循环回测每一个参数组合, 并输出每个参数组合的优化结果。普通模式下只能使用 CPU 的一个核。
- `runParallelOptimization(self, strategyClass, optimizationSetting)`: 运行多进程模式的优化参数, 也就是说并行运行多个进程回测每一个参数组合, 并输出每个参数组合的优化结果。
- `updateDailyClose(self, dt, price)`: 更新每日收盘价。
- `calculateDailyResult(self)`: 按照逐日盯市的方式计算每日的交易盈亏和持仓盈亏, 并汇总成最终按日统计的盈亏情况。
- `calculateDailyStatistics(self, df)`: 按照逐日盯市的方式统计每日的交易盈亏和持仓盈亏, 并且计算盈亏相关数据, 如总收益、年化收益、最大回撤、日均盈亏和夏普比率等。
- `showDailyResult(self, df=None, result=None)`: 显示按照逐日盯市方式统计出的回测结果, 并显示回测结果图, 包括资金子图、回撤子图、每日盈亏子图和每日盈亏分布子图。



## 4.5.2 参数优化设置

- `addParameter(self, name, start, end=None, step=None)`: 增加第一个优化参数 `atrLength`, 起始 12, 结束 20, 步进 2。
- `generateSetting(self)`: 生成优化参数组合。`vn.py` 目前仅仅实现的是穷举法, 先创建参数名列表, 使用迭代工具生产参数对组合, 再把参数对组合打包到由一个个字典组成的列表中。
- `setOptimizeTarget(self, target)`: 设置优化目标字段, 优化目标可以是 `totalNetPnl`, `maxDrawdown` 和 `sharpeRatio` 等。
- `optimize(strategyClass, setting, targetName, mode, startDate, initDays, endDate, slippage, rate, size, priceTick, dbName, symbol)`: 运行多进程模式的优化参数时, 每个进程中运行的函数。

## 4.5.3 调用回测和优化模块

在“`vnpy-1.9.0\examples\CtaBacktesting`”文件夹内有 3 个策略回测的示例文件, 分别如下所示。

- `backtesting_IF.ipynb`: 用于回测和优化 1 分钟级别沪深 300 股指期货品种。
- `backtesting_rb.ipynb`: 用于回测 1 分钟级别螺纹钢期货品种。
- `backtesting_portfolio.ipynb`: 用于对多品种进行回测。

该文件的后缀名为 `ipynb`, 故需要用 Jupyter Notebook 打开运行, 下面以 `backtesting_IF.ipynb` 为例, 讲述如何调用回测和优化模块的方法。

### 1. 导入库

`BacktestingEngine` 是策略回测引擎, `OptimizationSetting` 用于参数优化, `MINUTE_DB_NAME` 用于在 MongoDB 数据库中存放分钟级别数据, `AtrRsiStrategy` 是本次需要测试的策略。

```
%matplotlib inline
from vnpy.trader.app.ctaStrategy.ctaBacktesting import BacktestingEngine,
```



```
OptimizationSetting, MINUTE_DB_NAME
from vnpy.trader.app.ctaStrategy.strategy.strategyAtrRsi import AtrRsiStrategy
```

## 2. 创建回测引擎对象

创建回测引擎对象，方便以后被调用：

```
engine = BacktestingEngine()
```

## 3. 设置回测使用的数据

设置回测使用的数据为 K 线。在 MongoDB 数据库的分钟级别数据中找到 IF0000 的合约数据，然后设置开始回测的时间为 2015 年 1 月 1 日。

```
engine.setBacktestingMode(engine.BAR_MODE)    # 设置回测使用的数据为 K 线
engine.setDatabase(MINUTE_DB_NAME, 'IF0000')  # 设置使用的历史数据库
engine.setStartDate('20150101')              # 设置回测用的数据起始日期
```

## 4. 配置回测引擎参数

滑点设置为股指 1 跳，每一跳为 0.2；手续费为万分之 0.3；股指的合约大小是每点 300 元，即 300；股指合约的最小价格变动单位是 0.2；然后设置起始资金为 100 万元。

```
engine.setSlippage(0.2)    # 设置滑点为股指 1 跳
engine.setRate(0.3/10000)  # 设置手续费万分之 0.3
engine.setSize(300)        # 设置股指合约大小
engine.setPriceTick(0.2)   # 设置股指最小价格变动
engine.setCapital(1000000) # 设置回测本金
```

## 5. 在引擎中创建策略对象

d = {}用于在回测时直接修改参数，而不需要重新打开策略文件进行修改。  
Engine.initStrategy()用于创建策略对象，创建好后就可以直接运行回测了。

```
d = {'atrLength': 11}          # 策略参数配置
engine.initStrategy(AtrRsiStrategy, d)  # 创建策略对象
```



## 6. 运行回测

```
engine.runBacktesting() # 运行回测
```

runBacktesting()意味着开始进行策略回测，同时会打印出相关数据回放信息，如图 4-10 所示。

```
2018-08-06 15:01:09.708000    开始载入数据
2018-08-06 15:01:28.454000    载入完成，数据量：155070
2018-08-06 15:01:28.454000    开始回测
2018-08-06 15:01:28.504000    策略初始化完成
2018-08-06 15:01:28.504000    策略启动完成
2018-08-06 15:01:28.504000    开始回放数据
2018-08-06 15:01:39.480000    数据回放结束
```

图 4-10 数据回放信息

## 7. 显示逐日回测结果

```
engine.showDailyResult()
```

显示逐日回测统计的相关指标，如图 4-11 所示，然后展示的是 4 幅子图：资金子图、回撤子图、每笔交易盈亏子图和每笔盈亏分布子图，分别如图 4-12 到图 4-15 所示。

2018-08-06 15:02:11.279000	计算按日统计结果
2018-08-06 15:02:11.334000	
2018-08-06 15:02:11.334000	首个交易日： 2015-01-12
2018-08-06 15:02:11.334000	最后交易日： 2017-07-14
2018-08-06 15:02:11.334000	总交易日： 612
2018-08-06 15:02:11.334000	盈利交易日： 297
2018-08-06 15:02:11.334000	亏损交易日： 315
2018-08-06 15:02:11.334000	起始资金： 1000000
2018-08-06 15:02:11.334000	结束资金： 1,133,650.64
2018-08-06 15:02:11.335000	总收益率： 13.37%
2018-08-06 15:02:11.335000	年化收益： 5.24%
2018-08-06 15:02:11.335000	总盈亏： 133,650.64
2018-08-06 15:02:11.335000	最大回撤： -185,949.45
2018-08-06 15:02:11.335000	百分比最大回撤： -16.31%
2018-08-06 15:02:11.335000	总手续费： 148,769.36
2018-08-06 15:02:11.335000	总滑点： 267,420.0
2018-08-06 15:02:11.335000	总成交金额： 4,958,978,520.0
2018-08-06 15:02:11.335000	总成交笔数： 4,457.0
2018-08-06 15:02:11.336000	日均盈亏： 218.38
2018-08-06 15:02:11.336000	日均手续费： 243.09
2018-08-06 15:02:11.336000	日均滑点： 436.96
2018-08-06 15:02:11.336000	日均成交金额： 8,102,906.08
2018-08-06 15:02:11.336000	日均成交笔数： 7.28
2018-08-06 15:02:11.336000	日均收益率： 0.02%
2018-08-06 15:02:11.336000	收益标准差： 1.72%
2018-08-06 15:02:11.336000	Sharpe Ratio： 0.18

图 4-11 逐日回测统计的相关指标

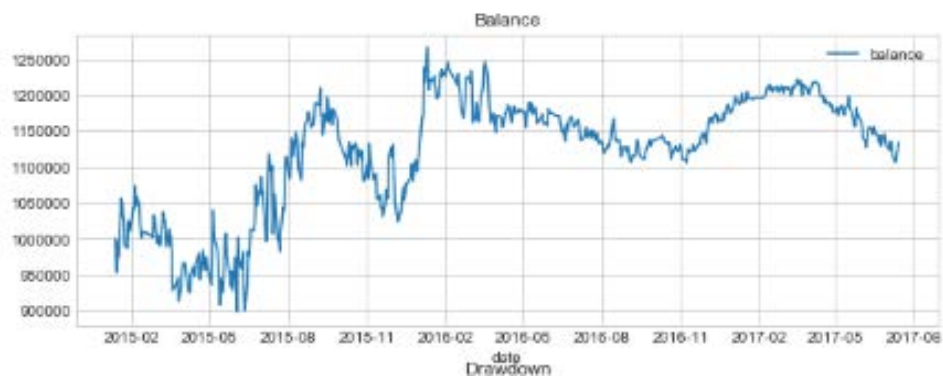


图 4-12 资金子图

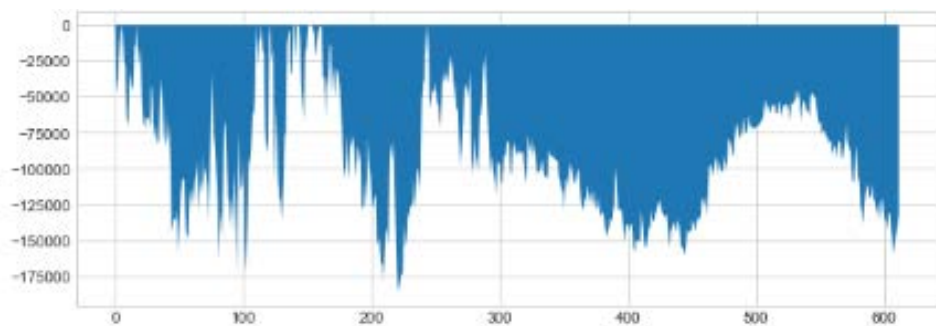


图 4-13 回撤子图



图 4-14 每笔交易盈亏子图

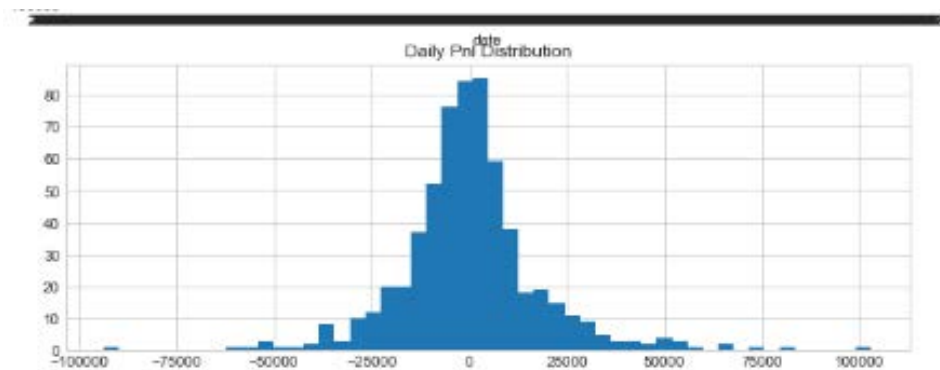


图 4-15 每笔盈亏分布子图

## 8. 显示逐笔回测结果

```
engine.showBacktestingResult()
```

同样显示的是回测相关指标和展示 4 幅子图。不同的是，其统计方式是按照“开仓-平仓”的方式进行逐笔统计的。

## 9. 显示前 10 条成交记录

成交记录会保存在成交字典（tradeDict）中，通过查询成交字典可以得到交易时间、多空方向、成交价格 and 成交数量等信息。

```
for i in range(10):
    d = engine.tradeDict[str(i+1)].__dict__
    print 'TradeID: %s, Time: %s, Direction: %s, Price: %s,
Volume: %s' % (d['tradeID'], d['dt'], d['direction'], d['price'], d['volume'])
```

其效果如图 4-16 所示

```
TradeID: 1, Time: 2015-01-12 09:18:00, Direction: 多, Price: 3550.0, Volume: 1
TradeID: 2, Time: 2015-01-12 09:27:00, Direction: 空, Price: 3529.8, Volume: 1
TradeID: 3, Time: 2015-01-12 09:35:00, Direction: 多, Price: 3540.8, Volume: 1
TradeID: 4, Time: 2015-01-12 10:02:00, Direction: 空, Price: 3536.2, Volume: 1
TradeID: 5, Time: 2015-01-12 10:03:00, Direction: 空, Price: 3541.4, Volume: 1
TradeID: 6, Time: 2015-01-12 13:34:00, Direction: 多, Price: 3492.2, Volume: 1
TradeID: 7, Time: 2015-01-12 13:35:00, Direction: 多, Price: 3491.6, Volume: 1
TradeID: 8, Time: 2015-01-12 14:14:00, Direction: 空, Price: 3478.8, Volume: 1
TradeID: 9, Time: 2015-01-12 14:15:00, Direction: 空, Price: 3470.2, Volume: 1
TradeID: 10, Time: 2015-01-12 14:19:00, Direction: 多, Price: 3496.8, Volume: 1
```

图 4-16 显示前 10 条成交记录

## 10. 优化配置

优化配置主要设置最大化夏普比率、最大化结束资金、最大化总赢利、最小化资金回撤等，这里示例优化的目标是最大总赢利。

优化参数可以设置多组，但是随着组数的增多，所需要的时间会呈指数地增加。利用 CPU 多进程优化（runParallelOptimization）在速度会有所提升，但是会导致电脑卡顿。当进行多维度的参数优化时，建立在云服务器上运行，因为它能够 24 小时流畅地“跑”策略，而不用担心在本机上由于卡顿或者其他操作而导致的宕机。

```
setting = OptimizationSetting()                # 新建一个优化任务设置对象
setting.setOptimizeTarget('totalNetPnl')        # 设置优化排序的目标是策略净赢利
setting.addParameter('atrLength', 12, 16, 2)    # 增加第一个优化参数 atrLength, 起始
                                                # 12, 结束 20, 步进 2
#setting.addParameter('atrMa', 20, 30, 5)       # 增加第二个优化参数 atrMa, 起始 20,
                                                # 结束 30, 步进 5
#setting.addParameter('rsiLength', 5)          # 增加一个固定数值的参数

# 执行多进程优化
import time
start = time.time()
resultList = engine.runParallelOptimization(AtrRsiStrategy, setting)
print u'耗时: %s' %(time.time()-start)
```

多进程优化会让 CPU 满载运行进而提高其运行速度，优化后结果如图 4-17 所示。

```
优化结果:
参数: {'atrLength': 16}, 目标: 265724.43959999647
参数: {'atrLength': 14}, 目标: 171968.15079999733
参数: {'atrLength': 12}, 目标: 163439.42939999666
```

图 4-17 优化后结果

## 11. 显示优化的所有统计数据

在运行完参数优化后，可以打印出每一组参数带来的回测效果，如年化收益、百分百最大回撤、夏普比率等。



```

for result in resultList:
    print '-' * 30
    print u'参数: %s, 目标: %s' %(result[0], result[1])
    print u'统计数据: '
    for k, v in result[2].items():
        print u'%s: %s' %(k, v)

```

其结果如图 4-18 所示。

```

参数: ['atrLength': 16], 目标: 265724.43959999847
统计数据:
startDate: 2015-01-12
endBalance: 1265724.4395999988
endDate: 2017-07-14
dailyTurnover: 7994311.568627451
annualizedReturn: 10.420566258823477
returnStd: 1.60432388835
totalTradeCount: 4401
maxDrawdown: -188182.96559999953
totalTurnover: 4892518680.0
profitDays: 295
sharpeRatio: 0.3693800419703181
maxDdPercent: -13.18594439633734
dailyReturn: 0.0382525028021
totalSlippage: 264060.0
dailyNetPnl: 434.19026078431125
dailyTradeCount: 7.1911764705882355
totalNetPnl: 265724.43959999847
totalDays: 612
lossDays: 317
dailySlippage: 431.47058823529414
dailyCommission: 239.82934705882352
totalReturn: 26.57244395999987
totalCommission: 146775.5604

```

图 4-18 显示优化的所有统计数据

# 第 5 章

## 经典 CTA 策略

本章主要介绍 vn.py 官方提供的经典 CTA 策略，这些策略代码都在“vnpy-1.9.0\vnpy\trader\app\ctaStrategy\strategy”文件夹内，内容主要包括策略原理、代码解释、策略回测和参数优化。

### 5.1 双均线策略

双均线策略是最简单的 CTA 策略，相当于编程中的“Hello World！”，其运用场景是日均中长线交易，掌握好双均线策略有利于理解更加复杂的交易策略。

#### 5.1.1 策略原理

对于每一个交易日，都可以计算出前  $N$  天的移动平均值，把这些移动平均值连成一条线，就叫作  $N$  日移动平均线。该策略用到两条均线，分别是快均线（5 天均线）和慢均线（20 天均线）。当快均线上升穿过慢均线时，形成金叉，给出买入信号。反之，当快均线下跌穿过慢均线时，形成死叉，给出卖出信号。如图 5-1 所示。



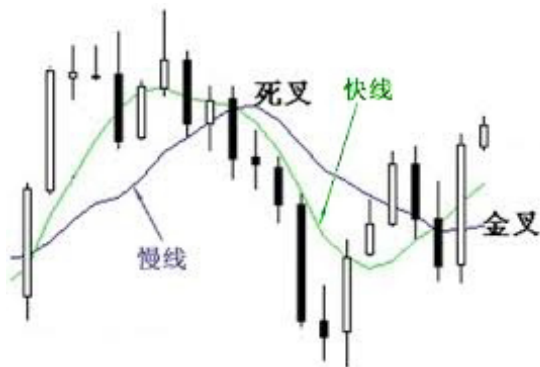


图 5-1 金叉和死叉形成

双均线策略的缺点是仅仅给出买卖信号，并无明确的过滤器和止损，故其稳定性不好。虽然使用该策略的玩家非常多，但是该策略已开始经逐渐失效了。

关于策略回测，在一般情况下，当拿到一组数据时：

第一步，用简单的向量回测方法来检测它是否有强的自相关性，如果数据在时间序列上表现出趋势，则适合使用 CTA 类策略。反之，如果表现出均值回归（即在一定价格区间上下震荡），则使用套利类策略。

第二步，当数据显示很强或者很弱的自相关性时，均需要更加精细化的策略回测，vn.py 提供了比较完整的策略回测模块，用户可以基于该模块进行数据回测。

## 5.1.2 向量回测

下面以上证 50ETF（510050）为例进行双均线的向量回测，其具体步骤如下。

（1）导入包。

```
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import tushare as ts
```

（2）用 Tushare 获取数据，然后用 sort\_index() 改变其时间排序方式，再计算



每天的价格变化，df.head(20)可以显示头 20 行数据，如图 5-2 所示。

```
# 获取数据
data = ts.get_hist_data('510050', '2017-01-01', '2018-01-01')
#50ETF 1 年日线历史数据

data = data.sort_index()
#按照日期排序

df = pd.DataFrame()
#创建新 DataFrame
df['close'] = data['close']
#收盘价
df['change'] = data['close'] - data['close'].shift(1)
#每日收盘价的变动
df.head(20)
#显示头 20 行数据（交易日）
```

date	close	change
2017-01-03	2.31	NaN
2017-01-04	2.33	0.02
2017-01-05	2.32	-0.01
2017-01-06	2.31	-0.01
2017-01-09	2.32	0.01
2017-01-10	2.31	-0.01
2017-01-11	2.30	-0.01

图 5-2 显示头 20 行数据（部分截取）

（3）计算两条均线，分别是 5 天移动均线和 20 天移动均线。  
rolling(window=5,center=False)意味着滚动回望窗口的周期是 5，不取中位数。

```
# 计算均值
df['ma5'] = df['close'].rolling(window=5, center=False).mean()
df['ma20'] = df['close'].rolling(window=20, center=False).mean()
df.head(20)
```

（4）从图 5-3 中可以观察到，ma5 是从第 5 个交易日才开始有的（统计的是前 4 天的收盘价，加上今天的收盘价），而 ma20 是从第 20 个交易日才开始产生的。所以到 2 月 6 日所有数据才完整起来，同时从这天起（第 20 个交易日）开始产生交易信号。



	close	change	ma5	ma20		close	change	ma5	ma20
date					date				
2017-01-03	2.31	NaN	NaN	NaN	2017-01-23	2.35	0.00	2.340	NaN
2017-01-04	2.33	0.02	NaN	NaN	2017-01-24	2.35	0.00	2.344	NaN
2017-01-05	2.32	-0.01	NaN	NaN	2017-01-25	2.36	0.01	2.348	NaN
2017-01-06	2.31	-0.01	NaN	NaN	2017-01-26	2.37	0.01	2.356	NaN
2017-01-09	2.32	0.01	2.318	NaN	2017-02-03	2.34	-0.03	2.354	NaN
2017-01-10	2.31	-0.01	2.318	NaN	2017-02-06	2.34	0.00	2.352	2.3305
2017-01-11	2.30	-0.01	2.312	NaN	2017-02-07	2.34	0.00	2.350	2.3320
2017-01-12	2.30	0.00	2.308	NaN	2017-02-08	2.35	0.01	2.348	2.3330
2017-01-13	2.31	0.01	2.308	NaN	2017-02-09	2.35	0.00	2.344	2.3345
2017-01-16	2.34	0.03	2.312	NaN	2017-02-10	2.37	0.02	2.350	2.3375
2017-01-17	2.33	-0.01	2.316	NaN	2017-02-13	2.38	0.01	2.358	2.3405
2017-01-18	2.34	0.01	2.324	NaN	2017-02-14	2.37	-0.01	2.364	2.3435
2017-01-19	2.33	-0.01	2.330	NaN	2017-02-15	2.37	0.00	2.368	2.3470
2017-01-20	2.35	0.02	2.338	NaN	2017-02-16	2.37	0.00	2.372	2.3505
					2017-02-17	2.36	-0.01	2.370	2.3530

图 5-3 从第 20 个交易日后开始产生信号

(5) 清除空值，从 2 月 6 日开始统计：

```
# 删除空值
df = df.dropna()
```

(6) 通过交易信号计算持仓情况。若快均线上穿慢均线，即  $ma5 > ma20$  时，产生买入信号。因为  $ma5$  和  $ma20$  是当天收盘的时候才刚刚统计出来的，所以其交易信号需要第二天开盘的时候执行。若是不注意写成当天执行的话，就隐含未来函数，导致策略回测结果偏高。因为函数必须满足时序不变性，即时间靠后的数据对时间靠前的结果不产生影响，未来函数就是时间靠后的数据对靠前结果有影响。

反之，若  $ma5 < ma20$ ，手头有多仓，则先平掉，然后再开相同数目的空仓，如图 5-4 所示。

```
# 持仓情况
df['pos'] = 0 # 初始化

#交易信号判断
df['pos'][df['ma5'] >= df['ma20']] = 10000
df['pos'][df['ma5'] < df['ma20']] = -10000
```

```
df['pos'] = df['pos'].shift(1).fillna(0)
df
```

	close	change	ma5	ma20	pos	pnl	fee	netpnl
date								
2017-02-06	2.34	0.00	2.352	2.3305	0.0	0.0	14.04	-14.04
2017-02-07	2.34	0.00	2.350	2.3320	10000.0	0.0	14.04	-14.04
2017-02-08	2.35	0.01	2.348	2.3330	10000.0	100.0	0.00	100.00
2017-02-09	2.35	0.00	2.344	2.3345	10000.0	0.0	0.00	0.00
2017-02-10	2.37	0.02	2.350	2.3375	10000.0	200.0	0.00	200.00
2017-02-13	2.38	0.01	2.358	2.3405	10000.0	100.0	0.00	100.00
2017-02-14	2.37	-0.01	2.364	2.3435	10000.0	-100.0	0.00	-100.00
2017-02-15	2.37	0.00	2.368	2.3470	10000.0	0.0	0.00	0.00
2017-02-16	2.37	0.00	2.372	2.3505	10000.0	0.0	0.00	0.00

图 5-4 通过交易信号计算持仓情况

(7) 计算净盈亏。

每天盈亏=当日持仓×当天价格变化

当仓位发生变化时，如做多变成做空，则会产生手续费。

手续费=开仓的总数（这里是 2 万股）×当日收盘价×手续费率（万分之三）

净盈亏=每天盈亏-手续费

如图 5-5 所示。

```
# 计算每日盈亏和手续费
df['pnl'] = df['pos']*df['change']

df['fee'] = 0 #初始化 fee
df['fee'][df['pos']!= df['pos'].shift(1)] = 20000 * df['close'] * 3 / 10000

df['netpnl'] = df['pnl'] - df['fee']
df
```



	close	change	ma5	ma20	pos	pnl	fee	netpnl
date								
2017-02-06	2.34	0.00	2.352	2.3305	0.0	0.0	14.04	-14.04
2017-02-07	2.34	0.00	2.350	2.3320	10000.0	0.0	14.04	-14.04
2017-02-08	2.35	0.01	2.348	2.3330	10000.0	100.0	0.00	100.00
2017-02-09	2.35	0.00	2.344	2.3345	10000.0	0.0	0.00	0.00
2017-02-10	2.37	0.02	2.350	2.3375	10000.0	200.0	0.00	200.00
2017-02-13	2.38	0.01	2.358	2.3405	10000.0	100.0	0.00	100.00
2017-02-14	2.37	-0.01	2.364	2.3435	10000.0	-100.0	0.00	-100.00
2017-02-15	2.37	0.00	2.368	2.3470	10000.0	0.0	0.00	0.00

图 5-5 计算净盈亏

(8) 计算累计盈亏, 如图 5-6 所示。

```
# 累计盈亏 cumsum()
df['cumpnl'] = df['netpnl'].cumsum()
df
```

	close	change	ma5	ma20	pos	pnl	fee	netpnl	cumpnl
date									
2017-02-06	2.34	0.00	2.352	2.3305	0.0	0.0	14.04	-14.04	-14.04
2017-02-07	2.34	0.00	2.350	2.3320	10000.0	0.0	14.04	-14.04	-28.08
2017-02-08	2.35	0.01	2.348	2.3330	10000.0	100.0	0.00	100.00	71.92
2017-02-09	2.35	0.00	2.344	2.3345	10000.0	0.0	0.00	0.00	71.92
2017-02-10	2.37	0.02	2.350	2.3375	10000.0	200.0	0.00	200.00	271.92
2017-02-13	2.38	0.01	2.358	2.3405	10000.0	100.0	0.00	100.00	371.92
2017-02-14	2.37	-0.01	2.364	2.3435	10000.0	-100.0	0.00	-100.00	271.92
2017-02-15	2.37	0.00	2.368	2.3470	10000.0	0.0	0.00	0.00	271.92
2017-02-16	2.37	0.00	2.372	2.3505	10000.0	0.0	0.00	0.00	271.92

图 5-6 计算累计盈亏

(9) 最后绘图得到累计盈亏曲线, 如图 5-7 所示。

```
df['cumpnl'].plot()
```

测试表明双均线策略并不理想。图中显示 2.3 万元人民币的本钱, 在 2017 年全年最终赢利 2 千元左右。

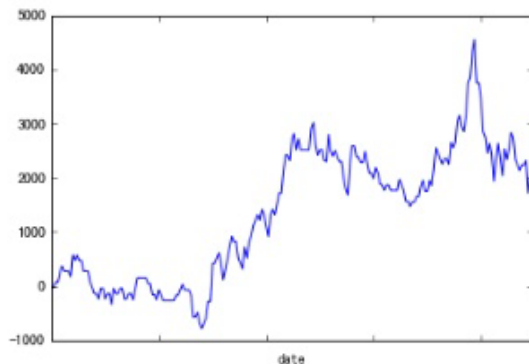


图 5-7 累计盈亏曲线

### 5.1.3 vn.py 回测

#### (1) 导入包。

pymongo 用于在 Python 环境中连接到 MongoDB 数据库，VtBarData 里保存了 K 线数据，DAILY\_DB\_NAME 是 vn.py 定义的 MongoDB 数据库存放日线数据的地方。

```
In[1]
#加载模块
%matplotlib inline
import tushare as ts
import pymongo                                #MongoDB 的 Python 客户端
from datetime import datetime

from vnpy.trader.vtObject import VtBarData    # vtObject 里面保存了所有交易平台用到的
                                              #数据
from vnpy.trader.app.ctaStrategy.ctaBase import DAILY_DB_NAME #加载 vn.py 数据库
                                              #常量
```

#### (2) 获取数据。

ts.get\_hist\_data 通过 Tushare 把行情数据导入内存中。Sort\_index()则可对时间序列进行重新排序，其顺序是从最老的数据到最新的数据，因为 Tushare 默认下

载的数据排序是从最新到最老。

```
In[2]
# 定义合约代码
Symbol = '510050'
Exchange = 'SSE' # 上交所
vtSymbol = '.'.join([symbol, exchange]) # 股票（合约代码+交易所代码）

# 下载历史数据
data = ts.get_hist_data(symbol, '2017-01-01', '2018-01-01')
data = data.sort_index()

print u'数据下载完成'
```

(3) 创建 MongoDB 连接。在 MongoClient 传入参数，第一个是 MongoDB 所在服务器的默认 IP（本机地址），第二个是默认端口 27017。然后返回客户端下的数据库对象，在数据库中查找 vtSymbol。

```
# 创建 MongoDB 链接
client = pymongo.MongoClient('localhost', 27017) # 默认 IP 和默认端口
collection = client[DAILY_DB_NAME][vtSymbol] # 数据库中查找 vtSymbol
collection.ensure_index('datetime') # 保证索引的唯一性
print u'MongoDB 链接成功'
```

在 MongoDB 数据库中可以看见连接已经创建好了，如图 5-8 所示。

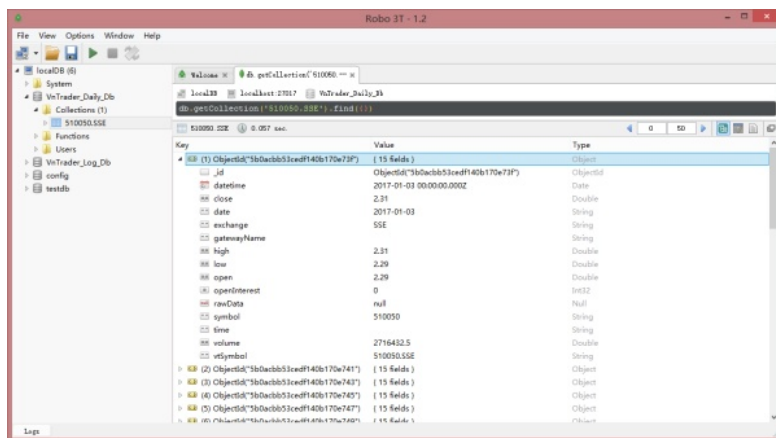


图 5-8 MongoDB 连接创建成功

(4) 将数据插入历史数据库。

注意下面代码的最后一行，`update_one()`保证数据库的唯一性（指这个键在已知的时候所对应的数值是唯一的，若重复插入则要么报错，要么新的值把老的值给覆盖掉）。`flt,{'$set':bar.__dict__}`出现的时候做一个更新。

`upsert=True` 意即当数据已经存在的时候去覆盖它，`upsert=False` 表示当数据已经存在的时候不做任何操作。

```
In[3]
# 将数据插入历史数据库
for row in data.iterrows():
    date = row[0]
    data = row[1]

    # 创建 K 线对象
    bar = VtBarData()
    bar.vtSymbol=vtSymbol          #品种代码+交易所
    bar.symbol=symbol              #品种代码
    bar.exchange=exchange          #交易所

    bar.date = date                #日期
    bar.datetime = datetime.strptime(date, '%Y-%m-%d') #把时间字符串解析为时间元组

    # K 线的高开低收数据
    bar.open = data['open']         #开盘价
    bar.high = data['high']         #最高价
    bar.low = data['low']           #最低价
    bar.close = data['close']       #收盘价
    bar.volume = data['volume']     #成交量

    flt={'datetime':bar.datetime}  #防止重复

    collection.update_one(flt,{'$set':bar.__dict__},upsert=True)

print u'数据插入完成'
```

(5) 导入策略模板文件中的策略模板对象。

```
In[4]
```



```

# 开发策略
import Numpy as np
from vnpy.trader.app.ctaStrategy.ctaTemplate import CtaTemplate #父类 CTA 策略模板

class DoubleMaStrategy(CtaTemplate):                                #子类双均线策略
    '''双均线策略 Demo'''
    className = 'DoubleMaStrategy'
    author = u'HSM'

    #策略参数
    initDays = 36 #数据是 21, 通过上面例子知道 2 月 6 日才能产生信号

    #策略变量
    barCount = 0
    closeArray = np.zeros(21) #同时计算新旧的 20 天均线, 所以需要 21 天数据
    ma5 = 0
    ma20 = 0
    LastMa5 = 0
    LastMa20 = 0

    #参数列表, 保存参数名称
    paramList=['name',
               'className',
               'author',
               'vtSmybol']

    #变量列表, 保存变量名称 (更多的是在 VnTrader 实盘运行 CTA 策略时, 在监控界面上显示刷新的字
    #段名)
    varList=['inited',
             'trading',
             'pos']

```

(6) 下面分别是构造函数、初始化策略、启动策略、停止策略和收到 Tick 推送。

```

#01-----
def __init__(self,ctaEngine, setting):
    '''构造函数'''
    super(DoubleMaStrategy,self).__init__(ctaEngine,setting)

    # 针对可变对象类型的变量, 需要在这里初始化
    self.closeArray = np.zeros(21)

```



```

#02-----
def onInit(self):
    '''初始化策略'''
    self.writeCtaLog(u'双均线策略初始化')

    initData = self.loadBar(self.initDays)

    for bar in initData:
        self.onBar(bar)

    self.putEvent()

#03-----
def onStart(self):
    '''启动策略'''
    self.writeCtaLog(u'双均线策略启动')
    self.putEvent()

#04-----
def onStop(self):
    '''停止策略'''
    self.writeCtaLog(u'双均线策略停止')
    self.putEvent()

#05-----
def onTick(self):
    '''收到行情 Tick 推送'''
    # 因为只是展示如何使用框架, 所以这里直接跳过, 实盘需要用户基于 Tick 数据自行合成 K 线
    pass

```

(7) 具体的双均线策略就在 onBar 函数上。

```

#06-----
def onBar(self, bar):
    '''收到 Bar 推送'''

    #缓存数据
    self.closeArray[0:20] = self.closeArray[1: 21]
    self.closeArray[-1]= bar.close

    self.barCount += 1

```



```

if self.barCount < self.initDays: #若还没过 36 天, 即插入 21 条日线, 直接返回
    return

#计算当日快慢均线和昨日快慢均线
self.ma5 = self.closeArray[16: 21].mean() #当日短均线
self.LastMa5 = self.closeArray[15: 20].mean() #昨日短均线
self.ma20 = self.closeArray[1: 21].mean() #当日长均线
self.LastMa20 = self.closeArray[0: 20].mean() #昨日长均线

# 判断买卖
crossOver = self.ma5 > self.ma20 and self.LastMa5 <= self.LastMa20
#金叉上穿
crossBelow = self.ma5 < self.ma20 and self.LastMa5 >= self.LastMa20
#死叉下穿

# 金叉和死叉的条件是互斥的。
# 所有的委托均以 K 线收盘价委托, 为了保证回测成交超价 5% 发单。
if crossOver:
    # 如果金叉时手头没有持仓, 则直接做多
    if self.pos == 0:
        self.buy(bar.close*1, 10000)
    # 如果有空头持仓, 则先平空, 再做多
    elif self.pos < 0:
        self.cover(bar.close*1, 10000)
        self.buy(bar.close*1, 10000)

#死叉相反
if crossBelow:
    if self.pos == 0:
        self.short(bar.close*1, 10000)
    elif self.pos > 0:
        self.sell(bar.close*1, 10000)
        self.short(bar.close*1, 10000)
self.putEvent()

```

(8) 因为是策略回测而不是实盘, 所以下面均可忽略。

```

#07-----
def onOrder(self, order):
    '''收到委托推送'''
    pass

```

```
#08-----
def onTrade(self,trade):
    '''收到成交推送'''
    pass

#09-----
def onStopOrder(self,so):
    '''停止单推送'''
    pass
```

(9) 在策略类设置完之后, 下面开始加载回测引擎。

```
In[5]
# 加载回测引擎
from vnpy.trader.app.ctaStrategy.ctaBacktesting import BacktestingEngine

# 创建回测引擎实例
engine = BacktestingEngine()

# 设置引擎的回测模式为 K 线
engine.setBacktestingMode(engine.BAR_MODE)

# 设置回测用的数据起始日期
engine.setStartDate('20170101', initDays=36)
engine.setEndDate('20180101')

# 设置产品相关参数
engine.setSlippage(0)          # 滑点设为 0
engine.setRate(3/10000)       # ETF 手续费
engine.setSize(1)             # ETF 每股为 1
engine.setPriceTick(0.001)    # ETF 最小价格变动
engine.setCapital(1)          # 为了只统计净盈亏, 设置初始资金为 1

# 设置使用的历史数据库
engine.setDatabase(DAILY_DB_NAME, vtSymbol)

# 在引擎中创建策略对象
engine.initStrategy(DoubleMaStrategy, {})

# 开始跑回测
engine.runBacktesting()
```



```
# 显示回测结果
engine.showDailyResult()
```

输出结果显示，从 2 月 6 日到 12 月 29 日，最终赢利 2 千多元人民币，最大回撤达 30%，夏普比率是 0.72，回测结果并不理想，如图 5-9 所示。这是纯双均线策略的特点，过于钝化，需要结合有效离场指标来降低亏损。

2018-09-09 23:15:30.846000	首个交易日:	2017-02-06
2018-09-09 23:15:30.847000	最后交易日:	2017-12-29
2018-09-09 23:15:30.847000	总交易日:	225
2018-09-09 23:15:30.847000	盈利交易日:	75
2018-09-09 23:15:30.847000	亏损交易日:	79
2018-09-09 23:15:30.847000	起始资金:	1
2018-09-09 23:15:30.847000	结束资金:	2,201.0
2018-09-09 23:15:30.847000	总收益率:	220,000.0%
2018-09-09 23:15:30.847000	年化收益:	234,666.67%
2018-09-09 23:15:30.847000	总盈亏:	2,200.0
2018-09-09 23:15:30.847000	最大回撤:	-2,800.0
2018-09-09 23:15:30.848000	百分比最大回撤:	-30,000.0%
2018-09-09 23:15:30.848000	总手续费:	0.0
2018-09-09 23:15:30.848000	总滑点:	0.0
2018-09-09 23:15:30.848000	总成交金额:	496,400.0
2018-09-09 23:15:30.848000	总成交笔数:	19.0
2018-09-09 23:15:30.848000	日均盈亏:	9.78
2018-09-09 23:15:30.848000	日均手续费:	0.0
2018-09-09 23:15:30.848000	日均滑点:	0.0
2018-09-09 23:15:30.848000	日均成交金额:	2,206.22
2018-09-09 23:15:30.848000	日均成交笔数:	0.08
2018-09-09 23:15:30.848000	日均收益率:	2.71%
2018-09-09 23:15:30.849000	收益标准差:	57.91%
2018-09-09 23:15:30.849000	Sharpe Ratio:	0.72

图 5-9 50ETF 回测逐日统计结果

最后是绘图结果，资金图显示赢利曲线从 2 月份一直升到 11 月份到达顶峰，然后大幅度回撤，第二幅回撤图显示在 9 月份和 12 月份期间都有比较大的回撤，如图 5-10 所示。图 5-11 所示的每笔盈亏分布图显示大部分交易日是盈亏接近零，极端情况较少。但是在极端情况下，大幅度亏损要多于大幅度赢利。

从双均线策略回测结果看来，`vn.py` 的回测并不具有优势，需要导入很多库，而且策略框架写起来很麻烦。但是，`vn.py` 回测模块相对于向量回测，第一个好处就是杜绝未来函数，而且对于要用到多个技术指标的通道突破策略，在已有框架下写起来显然比较轻松。



图 5-10 资金图和回撤图

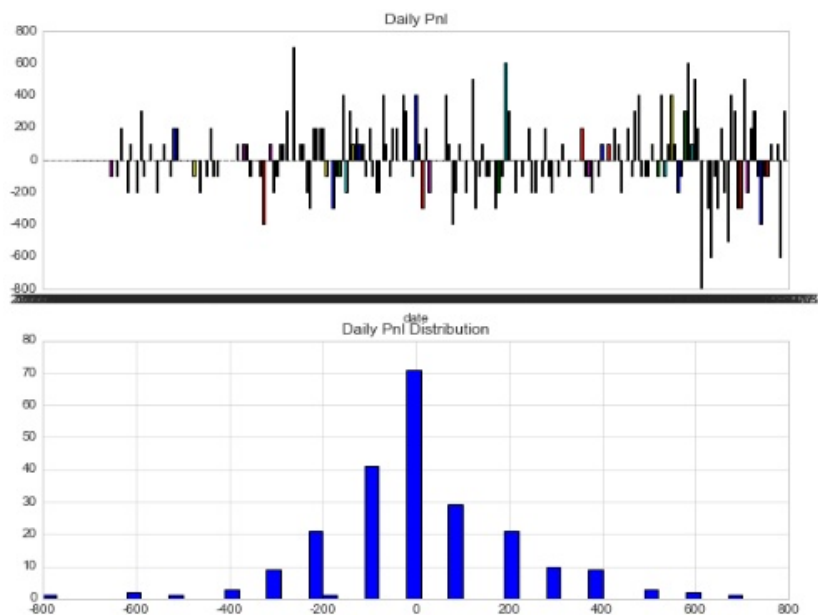


图 5-11 每笔交易盈亏图和每笔盈亏分布图

## 5.2 Dual Thrust 策略

### 5.2.1 策略原理

Dual Thrust 是一个趋势跟踪系统，由迈克尔·查莱克（Michael Chalek）在 20 世纪 80 年代开发，曾被 *Futures Truth* 杂志评为最赚钱的策略之一。Dual Thrust 系统具有简单易用、适用度广的特点，其思路简单、参数很少，配合不同的参数、止盈止损和仓位管理，可以为投资者带来长期稳定的收益，被投资者广泛应用于股票、货币、贵金属、债券、能源及股指期货市场等。在 Dual Thrust 策略中，对于震荡区间的定义非常关键，这也是该交易策略的核心和精髓。Dual Thrust 策略使用  $\text{Range} = \max(\text{HH} - \text{LC}, \text{HC} - \text{LL})$  来描述震荡区间的大小。其中 HH 是  $N$  日最高价的最高价，LC 是  $N$  日收盘价的最低价，HC 是  $N$  日收盘价的最高价，LL 是  $N$  日最低价的最低价。

（1）首先计算：①  $N$  日最高价的最高价 HH， $N$  日收盘价的最低价 LC；②  $N$  日收盘价的最高价 HC， $N$  日最低价的最低价 LL；③  $\text{Range} = \max(\text{HH} - \text{LC}, \text{HC} - \text{LL})$ ；④  $\text{BuyLine} = \text{Open} + k_1 \times \text{Range}$ ；⑤  $\text{SellLine} = \text{Open} + k_2 \times \text{Range}$ 。

（2）构造系统：①当价格向上突破上轨时，如果当时持有空仓，则先平仓，再开多仓；如果没有仓位，则直接开多仓；②当价格向下突破下轨时，如果当时持有多仓，则先平仓，再开空仓；如果没有仓位，则直接开空仓，如图 5-12 所示。

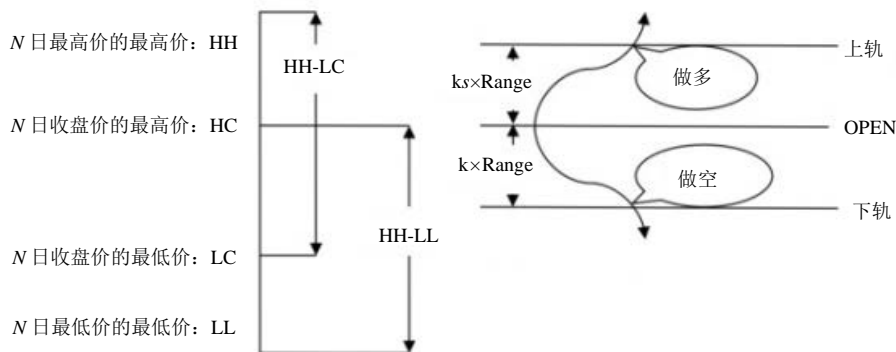


图 5-12 Dual Thrust 通道原理图

Dual Thrust 对于多头和空头的触发条件，考虑了非对称的幅度，即做多和做空参考的 Range 可以选择不同的周期数，也可以通过参数  $k1$  和  $k2$  来确定。

- 当  $k1 < k2$  时，多头相对容易被触发。
- 当  $k1 > k2$  时，空头相对容易被触发。

因此，在使用该策略时，一方面可以参考历史数据测试的最优参数；另一方面，则可以根据自己对后势的判断，或从其他大周期的技术指标入手，阶段性地动态调整  $k1$  和  $k2$  的值。

Dual Thrust 策略三大要素如下所示。

(1) 信号：当价格突破上下轨道的瞬间发出开仓信号。

(2) 过滤：除原始的通道过滤外，还有额外的条件来力求过滤震荡行情。

- 设置上下轨，若价格在轨道内波动，则不产生交易信号。
- 每天规定多开交易只做一次，避免在震荡行情中，日内不断开平仓操作从而损失手续费。
- 只当 Tick 行情推送合成的分钟 K 线高于日开盘价时判断交易方向为多头，设置突破的停止买入单，同理，分钟 K 线低于日开盘价时判断交易方向为空头，设置突破的停止卖出单（停止单的意思是在分钟 K 线内，条件触发时立刻进行开平仓操作）。

(3) 止损：固定点位止损策略，如多头仓位在价格下跌到下轨时自动平仓，空头仓位在价格突破到上轨时自动平仓。

## 5.2.2 策略代码解析

### 1. 设置策略参数

国内市场做多的远远多于做空，因此会表现出股价上升的时候慢吞吞，股价下跌时一旦突破阻力，其速度会相对迅猛。同时在股价下跌过程中，做多的玩家



为了止损会强行平仓，因此市场上多了很多空单，进一步加快下跌的速度。综上所述，价格向下跌的波动幅度要大于价格向上涨的幅度，故通道上轨的宽度（ $k1$ ）要小于下轨的宽度（ $k2$ ），所以设置  $k1 < k2$ 。

`fixedSize=1`，表示产生交易信号后，只交易 1 手。`initDays=10`，意味着数据初始化的天数为 10 天。

```
# 策略参数
fixedSize = 100
k1 = 0.4
k2 = 0.6

initDays = 10
```

## 2. 设置策略变量

50ETF 股指期货在时间上的相关性较差，市场参与者一般只关注前一天的数据而比较不关心过去几天的历史数据，所以这里对传统的 Dual Thrust 策略中  $N$  日取值为 1，即认为只有昨天股价的变化会对当天股价有影响。

Dual Thrust 策略需要用昨天的日 K 线数据，所以创建列表 `barList`，用于缓存每日 K 线的开盘价、最高价和最低价，还有 K 线波动（Range）、上下轨道和每天的收盘时间（股指期货默认是下午 14:55）。

`longEntered` 和 `shortEntered` 起着过滤器的作用，在 `onBar` 回调函数里面会用到。

```
# 策略变量
barList = []          # K 线对象的列表

dayOpen = 0
dayHigh = 0
dayLow = 0

range = 0
longEntry = 0
shortEntry = 0
exitTime = time(hour=14, minute=55)
```



```
longEntered = False  
shortEntered = False
```

### 3. onBar 函数

因为 Dual Thrust 策略是基于在通道突破的瞬间买入卖出的,即在 1 分钟 K 线内价格突破通道时触发条件进行交易,为了在追涨杀跌的过程中获得一个好的价格,所以在该 1 分钟内不保证成交。为了防止之前下的单子在上一个分钟没有成交,但是下一个分钟可能已经调整了价格,就用 `cancelAll()` 方法立刻撤销之前未成交的所有委托,保证策略在当前这 1 分钟开始时的整个状态是清晰和唯一的。

由于策略是由昨天和当日这两天的 K 线数据来产生交易信号的, `barList` 的长度设置为 2。用 `append(bar)` 把每一天 K 线数据插入到 `barList` 列表中,若该列表的长度小于或等于 2,会判断出数据缓存数量不足,则直接返回,不执行下面的任何操作了。当第 3 天到来时, `barList` 列表长度为 3, `if` 判断为 `False`,则会用 `pop(0)` 的方法来删除最老的数据,把最新的数据缓存下来。

```
def onBar(self, bar):  
    """收到 Bar 推送(必须由用户继承实现)"""  
    # 撤销之前发出的尚未成交的委托(包括限价单和停止单)  
    self.cancelAll()  
  
    # 计算指标数值  
    self.barList.append(bar)  
  
    if len(self.barList) <= 2:  
        return  
    else:  
        self.barList.pop(0)  
        lastBar = self.barList[-2]
```

当新的一天来临时,先对 `dayHigh` 做一个过滤,一般情况下第 1 天 `dayHigh` 为 0,从第 2 天起 `dayHigh` 不等于 0,有了昨天和今天的数据后,策略才可以真正运行,然后分别计算:

- 前 1 天的波动 `range`。



- 通道上轨（longEntry）。
- 通道下轨（shortEntry）。

缓存当天的开盘价、最高价和最低价，并且清空当天是否做出多空交易的状态（重新初始化）。若判断到当天还没有结束，必须用 max() 函数来更新 dayHigh 的数据从而得到当日的最高价，同理用 min() 函数来更新 dayLow 数据获取当天最低价。

```
# 新的一天
if lastBar.datetime.date() != bar.datetime.date():
    # 如果已经初始化
    if self.dayHigh:
        self.range = self.dayHigh - self.dayLow
        self.longEntry = bar.open + self.k1 * self.range
        self.shortEntry = bar.open - self.k2 * self.range

    self.dayOpen = bar.open
    self.dayHigh = bar.high
    self.dayLow = bar.low

    self.longEntered = False
    self.shortEntered = False
else:
    self.dayHigh = max(self.dayHigh, bar.high)
    self.dayLow = min(self.dayLow, bar.low)
```

对昨日波动 range 做一个过滤，range = 0 可能是策略刚刚跑到第一天的数据，也有可能昨天行情特殊（日 K 线最高价和最低价相等），还可能计算出了问题。当这些情况发生后直接返回，不进行下面的操作。

过滤完一些特殊情况后，若判断还没有到收盘时间，即 14: 55，可以考虑做一些开仓操作。但是注意，这里定义的 Dual Thrust 是一个日内趋势跟踪策略，到了收盘时间就必须平仓出场，不能出现隔夜持仓的情况。

开仓操作有三种情况：

- 手头上无仓位，若当前这 1 分钟 K 线的收盘价高于当日开盘价，则认为这是

一个向上的趋势。Dual Thrust 有个问题，就是标的价格在当日开盘价附近来回震荡，不断触碰上下轨道，会造成短时间里来回交易造成的亏损。解决方法就是设置一个过滤，每日不论空头还是多头操作，都只能进行 1 次交易。

`self.longEntered` 表示今天是否做过多仓操作，默认为 `False`。如果当天还没有做多的交易的话，就在通道上轨处挂上一个买入停止单（`stop=True`）。反之，若当前的分钟 K 线收盘价低于当日开盘价，则认为这是一个向下的趋势，同时判断到当天还没有做空交易的话，就在通道下轨处挂上一个卖出停止单。

- **手头上有多仓**，因为当天做了一次多头交易，先将多仓操作的状态更新为 `True`。又由于这是日内交易策略，接下来要做的就是设置平仓离场了。离场的位置是在通道下轨处下一个卖出停止单。同时，若判断当天还没有做空头交易，则除了平仓后还要反向做空。
- **手头上有空仓**，因为当天做了一次空头交易，所以先更新其状态为 `True`。然后设置平仓离场位置，这次的位置处于通道上轨，在这里挂一个买入停止单。同时，若判断当天还有一次多头交易的机会没用完，则需要反向开仓做多。

```
# 尚未到收盘
if not self.range:
    return

if bar.datetime.time() < self.exitTime:
    if self.pos == 0:
        if bar.close > self.dayOpen:
            if not self.longEntered:
                self.buy(self.longEntry, self.fixedSize, stop=True)
        else:
            if not self.shortEntered:
                self.short(self.shortEntry, self.fixedSize, stop=True)

    # 持有多头仓位
    elif self.pos > 0:
        self.longEntered = True

    # 多头止损
    self.sell(self.shortEntry, self.fixedSize, stop=True)

    # 空头开仓
```



```

        if not self.shortEntered:
            self.short(self.shortEntry, self.fixedSize, stop=True)

    # 持有空头仓位
    elif self.pos < 0:
        self.shortEntered = True

    # 空头止损
    self.cover(self.longEntry, self.fixedSize, stop=True)

    # 多头开仓
    if not self.longEntered:
        self.buy(self.longEntry, self.fixedSize, stop=True)

```

到了收盘时间，不管持有多仓还是空仓，用最快的速度平仓。为了保证平仓离场的速度和效果，这里不能用停止单，用的是限价单。当持有多仓时，限价卖单是低于市价单的 1%，同理，当持有空仓时，限价买单是高于市价单的 1%。

```

# 收盘平仓
else:
    if self.pos > 0:
        self.sell(bar.close * 0.99, abs(self.pos))
    elif self.pos < 0:
        self.cover(bar.close * 1.01, abs(self.pos))

# 发出状态更新事件
self.putEvent()

```

### 5.2.3 策略回测

在 Jupyter Notebook 中进行策略回测，vn.py 官方提供 3 套回测模板，分别是对沪深 300 股指期货、螺纹钢期货，以及品种组合的回测，在“vnpy-1.9.0\examples\CtaBacktesting”文件夹下。在同一文件夹下也有沪深 300 股指期货和螺纹钢期货 2010—2017 年的 1 分钟的历史数据，先把历史数据导入 MongoDB 数据库才有数据进行回测，2017 年以后的数据需要自行找数据源导入。

## 1. 设置回测使用的数据

首先由于是分钟级别数据，故回测模式是 K 线模式，其回测使用的数据是沪深 300 股指期货指数的 1 分钟数据，即 IF0000；然后设置回测用的数据起始和结束日期（setStartDate 和 setEndDate）。

```
engine.setBacktestingMode(engine.BAR_MODE)    # 设置引擎的回测模式为 K 线
engine.setDatabase(MINUTE_DB_NAME, 'IF0000')  # 设置沪深 300 股指期货 1 分钟 K 线数据
engine.setStartDate('20100416')              # 设置回测用的数据起始日期
engine.setEndDate('20180101')                 # 设置回测用的数据结束日期
```

## 2. 配置回测引擎参数

引擎参数包括交易滑点、手续费、合约规模、最小价格变动和起始资金。注意，每个期货品种都有各自的合约规模、手续费和最小价格变动，需要查询清楚再进行设置。

```
engine.setSlippage(0.2)      # 设置滑点为股指 1 跳
engine.setRate(0.5 / 10000)  # 设置手续费万分之 0.5
engine.setSize(300)          # 设置股指合约大小，IF 股指是 300 元/点
engine.setPriceTick(0.2)     # 设置股指最小价格变动，IF 股指最新价格变动单位是 0.2
engine.setCapital(1000000)    # 设置回测本金为 100 万元
```

运行完策略回测后，其结果如图 5-13 所示，尽管 Dual Thrust 策略稳稳地把握住了 2015 年末大牛市的行情，但在其他时间都是震荡市，假突破多，造成大部分时间的亏损，其回撤的规模也是巨大的，百分比最大回撤达-59.7%。但年化收益达 15 倍，这是因为原策略执行的手数是 100 手，即 fixedSize = 100，在下面的优化中需要把 fixedSize 改成 1 手。

同时，需要进一步改进该策略的过滤器来更好地避免在震荡行情做单，降低整个策略的风险。止损也可以考虑用固定百分数点位移动止损。



计算按日统计结果

首个交易日: 2010-04-26  
 最后交易日: 2017-12-29  
 总交易日: 1870  
 盈利交易日: 671  
 亏损交易日: 688  
 起始资金: 1000000  
 结束资金: 123,789,171.1  
 总收益率: 12,278.92%  
 年化收益: 1,575.9%  
 总盈亏: 122,789,171.1  
 最大回撤: -26,575,098.0  
 百分比最大回撤: -69.7%  
 总手续费: 13,960,828.9  
 总滑点: 18,528,000.0  
 总成交金额: 2.77216578e+11  
 总成交笔数: 3,088.0  
 日均盈亏: 65,662.66  
 日均手续费: 7,412.21  
 日均滑点: 9,908.02  
 日均成交金额: 148,244,159.36  
 日均成交笔数: 1.65  
 日均收益率: 0.26%  
 收益标准差: 4.58%  
 Sharpe Ratio: 0.88

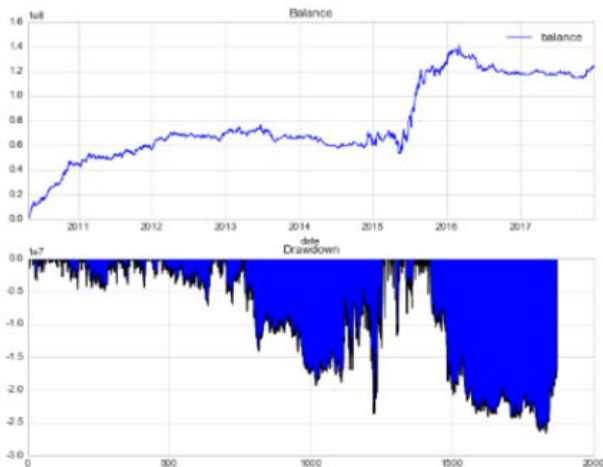


图 5-13 原始 Dual Thrust 策略回测效果

## 5.2.4 策略优化

具体步骤如下所述。

- (1) 修改执行手数, `fixedSize = 1`。
- (2) 把基于 1 分钟 K 线产生交易信号换成基于 5 分钟判断。

```

#-----
def __init__(self, ctaEngine, setting):
    """Constructor"""
    super(DualThrustStrategy, self).__init__(ctaEngine, setting)
    self.bg = BarGenerator(self.onBar, 5, self.onXminBar) # 创建K线合成器对象
    .....

#-----
def onBar(self, bar):
    """收到Bar推送(必须由用户继承实现)"""
    self.bg.updateBar(bar)

#-----
def onXminBar(self, bar):
    """收到x分钟K线"""
    #主逻辑判断从 onBar 函数转到 onXminBar 中
    .....
  
```

(3) 设置固定点位 (0.8%) 移动止损。

```
#策略变量
intraTradeHigh = 0          # 持仓期内的最高点
intraTradeLow = 0          # 持仓期内的最低点
longStop = 0               # 多头止损
shortStop = 0              # 空头止损
#策略参数
trailingPercent = 0.8      # 百分比移动止损
.....
#-----
def onXminBar(self, bar):
.....
    # 持有多头仓位
    elif self.pos > 0:
        self.longEntered = True

        # 计算多头持有期内的最高价, 以及重置最低价
        self.intraTradeHigh = max(self.intraTradeHigh, bar.high)
        self.intraTradeLow = bar.low

        # 计算多头移动止损
        longStop = self.intraTradeHigh * (1 - self.trailingPercent / 100)

        # 发出本地止损委托
        self.sell(longStop, self.fixedSize, stop=True)

        # 空头开仓单
        if not self.shortEntered:
            self.short(longStop, self.fixedSize, stop=True)

    # 持有空头仓位
    elif self.pos < 0:
        self.shortEntered = True

        self.intraTradeLow = min(self.intraTradeLow, bar.low)
        self.intraTradeHigh = bar.high

        shortStop = self.intraTradeLow * (1+self.trailingPercent / 100)
        self.cover(shortStop, self.fixedSize, stop=True)
```



```
# 多头开仓单
if not self.longEntered:
    self.buy(shortStop, self.fixedSize, stop=True)
.....
```

(4) 通过优化参数, 得到新的  $k_1$ ,  $k_2$  数据, 即  $k_1 = 0.4$ ,  $k_2 = 0.7$ 。

```
setting = OptimizationSetting()
setting.setOptimizeTarget('sharpeRatio')
setting.addParameter('k1', 0.2, 0.7, 0.1)
setting.addParameter('k2', 0.2, 0.8, 0.1)

# 执行多进程优化
import time
start = time.time()
resultList = engine.runParallelOptimization(DualThrustStrategy, setting)
print u'耗时: %s' %(time.time() - start)
```

# 新建一个优化任务设置对象  
# 设置优化排序的目标是夏普比率  
# 增加第一个优化参数  $k_1$ , 起始 0.2, 结  
# 束 0.7, 步进 0.1  
# 增加第二个优化参数  $k_2$ , 起始 0.2, 结  
# 束 0.8, 步进 0.1

(5) 优化后结果: 策略改进后, 百分比最大回撤降低至-6.16%, 夏普比率升至 1.99, 如图 5-14 所示。

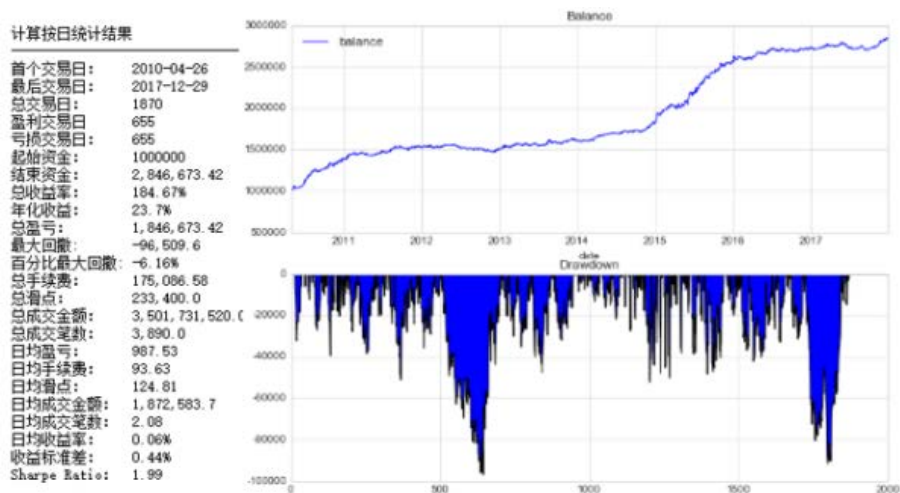


图 5-14 改进版 Dual Thrust 回测结果



### 5.2.5 滚动回测

以 3 年为滚动周期，步进是半年，对 20100416 至 20180101 区间进行滚动回测及参数优化，并且收集表现最好的前 3 组优化参数。

通过对历史优化参数的统计分析来预测未来的优化参数。其选择标准如下：

- 近年参考权重大于远年。
- 若优化参数整体变化不大，则取众数。
- 若优化参数整体发生变化，则在近年区间取众数。
- 若前一年参数发生剧烈变化，则对该区间进行更为细致的滚动回测和参数优化。

图 5-15 展示其历史优化参数整体变化不大，故取众数，即  $k1 = 0.4$ ， $k2 = 0.7$ 。同时，优化参数变化不大也说明策略稳健。

回测区间	最优参数		次优参数		第三优参数	
	k1	k2	k1	k2	k1	k2
20100416--20130601	0.5	0.6	0.5	0.7	0.4	0.6
20110101--20140101	0.5	0.6	0.5	0.7	0.5	0.8
20110601--20140601	0.5	0.8	0.5	0.7	0.5	0.6
20120101--20150101	0.5	0.7	0.5	0.8	0.4	0.7
20120601--20150601	0.5	0.8	0.4	0.8	0.4	0.7
20130101--20160101	0.4	0.7	0.4	0.6	0.5	0.7
20130601--20160601	0.4	0.7	0.4	0.6	0.5	0.7
20140101--20170101	0.4	0.7	0.4	0.6	0.5	0.7
20140601--20170601	0.4	0.6	0.4	0.7	0.5	0.6
20150101--20180101	0.4	0.6	0.4	0.7	0.5	0.6
2018至今	0.4	0.7				

图 5-15 历史优化参数

预测优化参数历史表现为年化收益达 23.7%，百分比最大回撤是-6.16%，夏普比率达 1.99，如图 5-16 所示。



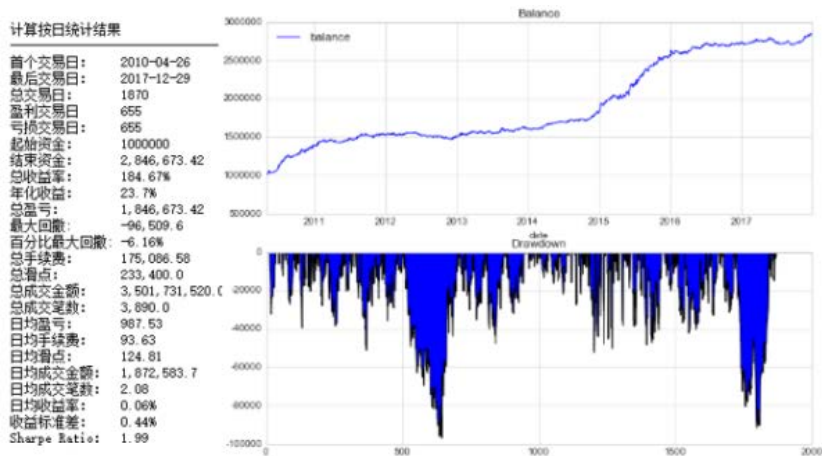


图 5-16 预测优化参数历史表现

对 2018 年以后的预测效果：2018 年春季发生较大的回撤，百分比最大回撤达 -7.15%，之后策略表现好转，年化收益为 13.64%，夏普比率达 1.02，如图 5-17 所示。

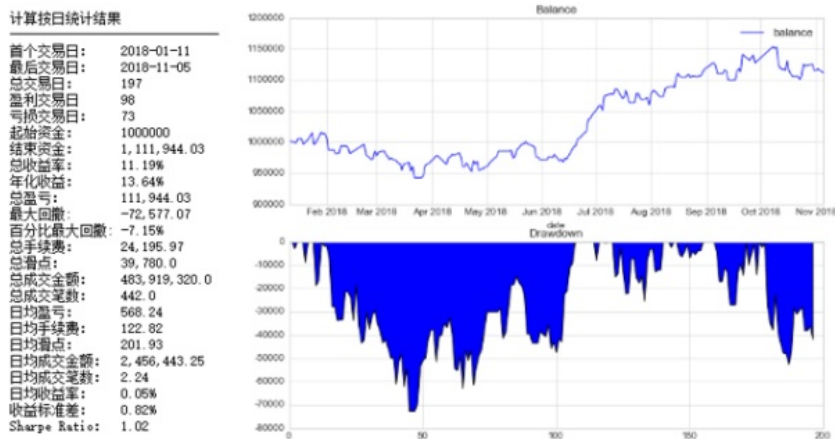


图 5-17 对 2018 年以后的预测效果

## 5.3 AtrRsi 策略

AtrRsi 策略主要依靠两个技术指标生成交易信号，具体如下所述。

## 5.3.1 ATR 指标

### 1. ATR 指标原理

平均真实波动范围（Average True Range），简称 ATR 指标，是由威尔斯·韦尔德（J. Welles Wilder）发明的。ATR 指标主要是用来衡量市场波动的强烈度，即用来显示市场变化率的指标。注意，这一指标主要用来衡量价格的波动，并不能直接反映价格走向及其趋势稳定性。

这一指标对于长期持续边幅移动的时段是非常典型的，这一情况通常发生在市场的顶部，或者是在价格巩固期间。该指标值越高，趋势改变的可能性就越大；该指标值越低，趋势改变的可能性就越小。当 ATR 线上升时，意味着资产的波动性在增加；当 ATR 线下降时，意味着资产的波动性在减少。ATR 不会显示资产移动的方向。

图 5-18 展示了 ATR 值是如何来表示波动性的高低的。

- 区间①的 ATR 值较高，表明波动性较高。
- 区间②的 ATR 值较低，表明波动性较低。



图 5-18 ATR 值与波动性的关系



## 2. ATR 指标计算

首先应计算出 TR（即当天的真实波幅），如图 5-19 所示，#2，#3 为市场出现跳空高开和跳空低开的情况。TR 在当日最高价与最低价，当日最高价与昨日收盘价，当日最低价与昨日收盘价这 3 种情况中取最大值。

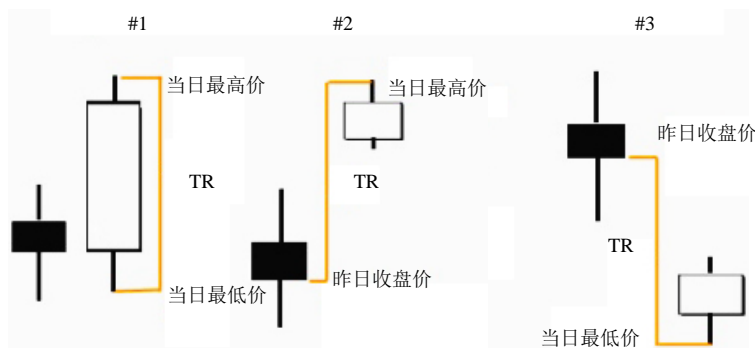


图 5-19 计算 ATR 所考虑的 3 种情况

由于一天的 TR 缺乏效率及代表性，韦尔德用 ATR 来更好地衡量市场的波动性；一般而言，市场常用的数据周期是 14 及 21，这意味着如果投资者在日图看 ATR，14 = 14 天；如果是在周图看 ATR，14 = 14 周。ATR 的计算公式：

$$TR_t = \max[\text{high}_t - \text{low}_t, \text{abs}(\text{high}_t - \text{close}_{t-1}), \text{abs}(\text{low}_t - \text{close}_{t-1})]$$

$$\text{ATR} = (\text{前 13 天的 TR} + \text{当天的 TR}) / 14$$

## 3. ATR 指标信号判断

除了通过用 ATR 数值大小直接判断市场波动性大小外，也可以通过对比当天平均真实波幅（ATR）和过去  $N$  天平均波幅（ATRMa）来判断市场波动性趋势。

- 若  $\text{ATR} > \text{ATRMa}$ ，则说明市场波动性增大，趋势正在增强。
- 若  $\text{ATR} < \text{ATRMa}$ ，则说明市场波动性减小，趋势开始减弱。

## 5.3.2 RSI 指标

### 1. RSI 指标原理

相对强弱指标 (Relative Strength Index)，简称 RSI 指标，也是由威尔斯·韦尔德发明的。RSI 指标是根据市场上供求关系平衡的原理，通过比较一段时期内单个标的物价格或整个市场指数的涨跌幅度来分析判断市场上多空双方买卖力量的强弱程度，从而判断未来市场走势的技术指标。

简单来说，RSI 是一种用来评估“买卖盘双方力道强弱”情况的技术指标，买家是代表金钱的力量，卖家是代表持货的力量。若买方力量稍逊，价格就会向下发展；相反，若卖方力量不足，价格就会向上发展。

### 2. RSI 指标计算

RSI 指标的计算公式：

$$\text{强弱指标RSI} = 100 - \frac{100}{1 + \text{RS}}$$

$$\text{相对强弱RS} = \frac{x\text{天内收盘涨数和的平均值}}{x\text{天内收盘跌数和的平均值}}$$

### 3. RSI 指标信号判断

一般而言，RSI 以 50 为中界线，大于 50 视为多头行情，小于 50 视为空头行情，如图 5-20 所示。



图 5-20 RSI 超买超卖区间

- 当 RSI>70 时，属于超买状态，后续行情有可能出现回调或转势，应该卖出；

- 当  $RSI < 30$  时，属于超卖状态，短期反弹概率较高，建议买入。

#### 4. RSI 指标的缺点

RSI 指标在高档或低档有时会有钝化的现象，因此会发生过早卖出或买进的情况。

RSI 只能作为一个警告信号，并不意味着市场必然朝这个方向发展，尤其在市场剧烈震荡时，超卖还是超买必须参考其他指标综合分析，如利用长天期的 RSI 均线与 RSI 线的关系来进行买卖信号判断，不能单独依赖 RSI 的信号而做出买卖决定。

背离走势的信号通常都是事后历史，而且有背离走势发生之后，行情并无反转的现象。有时背离一两次才真正反转，因此这方面的研判必须不断分析历史资料以积累经验。

由于 RSI 是一种比率的指标，因此在趋势分析的能力上会较弱。

盘势进入横盘整理时，长短天期的 RSI 也容易形成重复交叉的情形。

### 5.3.3 策略原理

该策略只用到两个技术指标。ATR 指标用于过滤，当  $ATR > ATRMa$  时，显示市场波动性增大，趋势正在增强。只有在市场出现趋势的时候做单（追涨杀跌），赢利的机会才会增大。RSI 指标则用于产生交易信号，当  $RSI >$  规定上限时，开仓做多；反之，当  $RSI <$  规定下限时，开仓做空。开仓之后就需要考虑如何赢利离场或者止损离场，这个策略用的是固定百分比点位的移动止损。例如，70 这个点位开仓后，行情一路走高至日高点 100，移动止损设了 99%，当行情开始回落时，会在 99 这个点位自动平仓离场，从而把握住了 29 点的赢利。

AtrRsi 策略三大要素如下所示。

（1）信号：RSI 指标。

(2) 过滤: ATR 指标。

(3) 出场: 固定百分百点位移动止损。

### 5.3.4 策略代码解析

#### 1. 策略参数

ATR 指标初始化需要用到 11 天数据, 即 11 天以后才能产生 ATR 值; 而 ATR 移动平均 (ATRMa) 则是统计 10 天内的平均值。

RSI 指标规定初始化需要用到 5 天数据, 16 是开仓的阈值。当  $RSI > 50 + 16$ , 即  $RSI > 66$  时, 开仓做多; 当  $RSI < 34$  时, 开仓做空。这里并不用市场公认的 RSI 高于 70 做多, RSI 低于 30 做空, 因为用的人多了, 交易信号已经失效。

fixedSize = 1 意味着每次开平仓操作只交易 1 手。

```
# 策略参数
atrLength = 11          # 计算 ATR 指标的窗口数
atrMaLength = 10        # 计算 ATR 均线的窗口数
rsiLength = 5           # 计算 RSI 的窗口数
rsiEntry = 16           # RSI 的开仓信号
trailingPercent = 0.8   # 百分比移动止损
initDays = 10           # 初始化数据所用的天数
fixedSize = 1           # 每次交易的数量
.....
#-----
def onInit(self):
    """初始化策略 (必须由用户继承实现)"""
    self.writeCtaLog(u'%s 策略初始化' % self.name)
    # 初始化 RSI 入场阈值
    self.rsiBuy = 50 + self.rsiEntry
    self.rsiSell = 50 - self.rsiEntry
.....
```

#### 2. onBar 函数

在开仓交易前, 为了防止之前下的挂单在上 1 分钟没有成交, 但是下 1 分钟



可能已经调整了价格，就用 `cancelAll()` 方法立刻撤销之前未成交的所有委托，保证策略在当前这 1 分钟开始的时候整个状态是清晰和唯一的。

对于本地化 K 线管理模块( `am = self.am`, 本地化的作用是节省不必要的缓存, 提高计算速度), 若检测到 K 线管理模块还没有插入足够多的数据, 即初始化状态为 `False`, 则直接返回不进行下面的操作; 反之则计算 ATR 指标、ATR 移动平均和 RSI 指标。

```
def onBar(self, bar):
    """收到 Bar 推送(必须由用户继承实现)"""
    self.cancelAll()

    # 保存K线数据
    am = self.am
    am.updateBar(bar)
    if not am.inited:
        return

    # 计算指标数值
    atrArray = am.atr(self.atrLength, array=True)
    self.atrValue = atrArray[-1]
    self.atrMa = atrArray[-self.atrMaLength:].mean()

    self.rsiValue = am.rsi(self.rsiLength)
```

- **当前无仓位**, 先初始化 K 线日高点和日低点的值, 用的是当前 K 线的最高价和最低价。若 ATR 指标往上走超过了移动平均, 说明趋势正在增强, 市场波动越来越大, 则用 RSI 指标来开仓操作了。RSI 的判断标准是当其指标高于 66 时, 发出买入委托, 用高于市价 5 个点来保证能够成交, 委托数量为 1 (fixedSize)。若 RSI 低于其卖出阈值, 则用低于市价 5 个点来保证成交。
- **当前持有多仓**, 用 `max()` 函数来统计当日 K 线达到的最高点, 日低点则直接更新当前 K 线最低价。移动止损设置为: 当 K 线达到日高点后, 回落 0.8% 时用停止单 (Stop Order) 进行平仓离场。
- **当前持有空仓**, 用 `min()` 函数来统计当日 K 线达到的最低点, 日高点则是直接更新当前 K 线最高价。移动止损设置为当 K 线达到日低点后, 反弹 0.8% 时用



停止单（Stop Order）进行平仓离场。

```
# 判断是否要进行交易

# 当前无仓位
if self.pos == 0:
    self.intraTradeHigh = bar.high
    self.intraTradeLow = bar.low

    # ATR 数值上穿其移动平均线, 说明行情短期内波动加大
    # 即处于趋势的概率较大, 适合 CTA 开仓
    if self.atrValue > self.atrMa:
        # 使用 RSI 指标判断趋势行情时, 会在超买超卖区钝化特征, 作为开仓信号
        if self.rsiValue > self.rsiBuy:
            # 这里为了保证成交, 选择超价 5 个点下单
            self.buy(bar.close+5, self.fixedSize)

        elif self.rsiValue < self.rsiSell:
            self.short(bar.close-5, self.fixedSize)

# 持有多头仓位
elif self.pos > 0:
    # 计算多头持有期内的最高价, 以及重置最低价
    self.intraTradeHigh = max(self.intraTradeHigh, bar.high)
    self.intraTradeLow = bar.low

    # 计算多头移动止损
    longStop = self.intraTradeHigh * (1-self.trailingPercent/100)

    # 发出本地止损委托
    self.sell(longStop, abs(self.pos), stop=True)

# 持有空头仓位
elif self.pos < 0:
    self.intraTradeLow = min(self.intraTradeLow, bar.low)
    self.intraTradeHigh = bar.high

    shortStop = self.intraTradeLow * (1+self.trailingPercent/100)
    self.cover(shortStop, abs(self.pos), stop=True)

# 同步数据到数据库
self.saveSyncData()
```



```
# 发出状态更新事件
self.putEvent()
```

### 5.3.5 策略回测

回测相关数据设置, 测试时间是 2011—2018 年, 测试的品种是沪深 300 股指期货指数数据 (IF0000), 故引擎参数与 Dual Thrust 策略的设置一样。

```
# 设置回测使用的数据
engine.setBacktestingMode(engine.BAR_MODE)      # 设置引擎的回测模式为 K 线
engine.setDatabase(MINUTE_DB_NAME, 'IF0000')    # 设置 IF 股指期货
engine.setStartDate('20110416')                # 设置回测用的数据起始日期
engine.setEndDate('20180101')                  # 设置回测用的数据结束日期

# 配置回测引擎参数
engine.setSlippage(0.2)                        # 设置滑点为股指 1 跳
engine.setRate(0.5 / 10000)                   # 设置手续费为万分之 0.5
engine.setSize(300)                           # 设置股指合约大小
engine.setPriceTick(0.2)                      # 设置股指最小价格变动单位
engine.setCapital(1000000)                     # 设置回测本金
```

初始本金为 100 万元人民币, 期末本金为 199 万元人民币, 年化收益为 12.8%, 百分比最大回撤达 -17.33%, 夏普比率达 0.6, 但是在 2016 年后表现不理想, 存在持续亏损, 如图 5-21 所示。

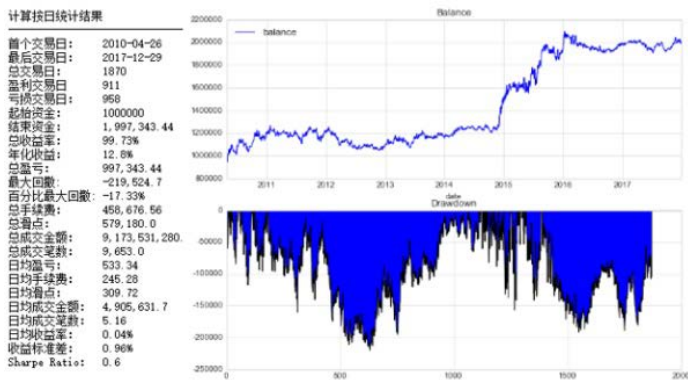


图 5-21 原始 AtrRsi 策略回测结果

## 5.3.6 滚动回测

### 1. 参数优化设置

本次参数优化的目标是夏普比率,需要优化的参数有 3 个维度,分别计算 ATR 指标所需要的回望窗口数 (artLength)、ATR 平均的回望窗口数 (atrMaLength) 和 RSI 指标的回望窗口数 (rsiLength)。其构建的优化组合为  $5 \times 4 \times 3 = 60$  个,故所需要的时间较长,若再增加一个维度的参数,优化时间就会呈指数级增长。

```
# 优化配置
setting = OptimizationSetting()           # 新建一个优化任务设置对象
setting.setOptimizeTarget('sharpeRatio') # 设置优化排序的目标是夏普比率
setting.addParameter('atrLength', 18, 30, 2) # 增加第一个优化参数 atrLength, 起始
                                             # 12, 结束 20, 步进 2
setting.addParameter('atrMaLength', 6, 14, 2) # 增加第二个优化参数 atrMa, 起始 20, 结
                                             # 束 30, 步进 5
setting.addParameter('rsiLength', 4, 8, 1)   # 增加第三个优化参数 rsiLength, 起始 4,
                                             # 结束 8, 步进 1

# 执行多进程优化
import time
start = time.time()
resultList = engine.runParallelOptimization(AtrRsiStrategy, setting)
print u'耗时: %s' %(time.time()-start)
```

### 2. 滚动回测设置

以 3 年为滚动周期,步进是半年,对 20100416 至 20180101 区间进行滚动回测及参数优化,并且收集表现最好的前 3 组优化参数。

通过对历史优化参数的统计分析,预测未来的优化参数。其选择标准如下:

- 近年参考权重大于远年。
- 若优化参数整体变化不大,则取众数。
- 若优化参数整体发生变化,则在近年区间取众数。
- 若前一年参数发生剧烈变化,则对该区间进行更为细致的滚动回测和参数优化。



如图 5-22 所示，历史优化参数整体发生变化，故在近年区间取众数，即  $\text{atrLength} = 22$ ， $\text{atrMaLength} = 12$ ， $\text{rsiLength} = 7$ 。

回溯区间	最优参数			次优参数			第三优参数		
	atrLength	atrMaLength	rsiLength	atrLength	atrMaLength	rsiLength	atrLength	atrMaLength	rsiLength
20100416--20130601	30	6	7	22	6	7	20	6	7
20110101--20140101	20	8	7	30	6	7	22	6	7
20110601--20140601	20	8	7	30	6	7	24	8	7
20120101--20150101	20	10	7	22	8	7	20	10	7
20120601--20150601	22	8	7	24	12	7	22	8	7
20130101--20160101	30	6	7	28	6	7	28	10	7
20130601--20160601	30	6	7	28	6	7	24	12	7
20140101--20170101	22	12	5	24	12	5	30	10	6
20140601--20170601	22	12	7	24	12	7	30	10	7
20150101--20180101	22	12	5	24	12	5	28	12	5
2018至今	22	12	7						

图 5-22 历史优化参数

预测优化参数历史表现是年化收益为 11.72%，百分比最大回撤为-18.89%，夏普比率达 0.58，如图 5-23 所示。

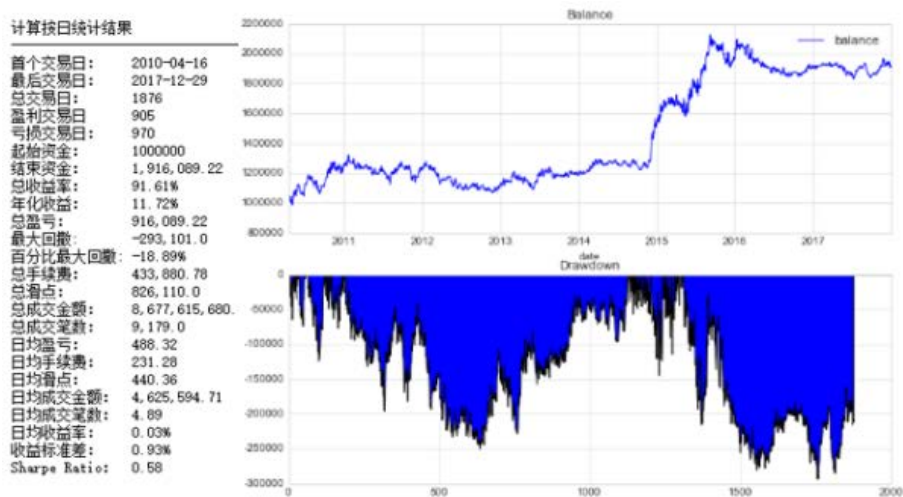


图 5-23 预测优化参数历史表现

对 2018 年以后的预测效果为年化收益提升到 37.42%，百分比最大回撤为 -6.61%，夏普比率达 1.62，如图 5-24 所示。

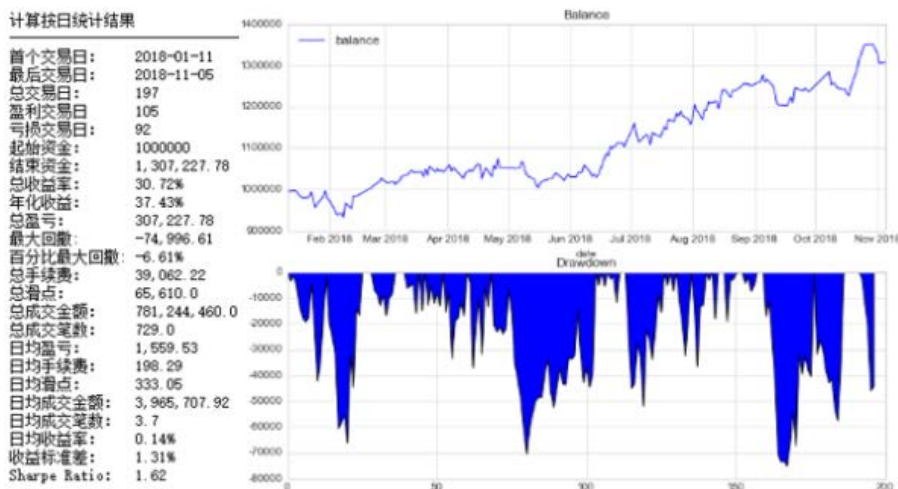


图 5-24 对 2018 年以后的预测效果

## 5.4 金肯特纳通道策略

### 5.4.1 策略原理

金肯特纳通道策略是一个典型的通道突破策略，即当价格突破通道上轨时做多，当价格走低突破通道下轨时做空，如图 5-25 所示。轨道计算的思路也相对简单，先计算移动均线（MA），并且统计 ATR 指标，设置一定的通道宽度偏差  $X$ ，如下：

- 上轨 =  $MA + X \times ATR$
- 下轨 =  $MA - X \times ATR$

因为 ATR 指标相对于标准差能够捕捉到 K 线跳空高开或者跳空低开的情况，所以更适合于一些在短期内有较大波动的品种，如股指期货或者有“小股指”之称的螺纹钢。

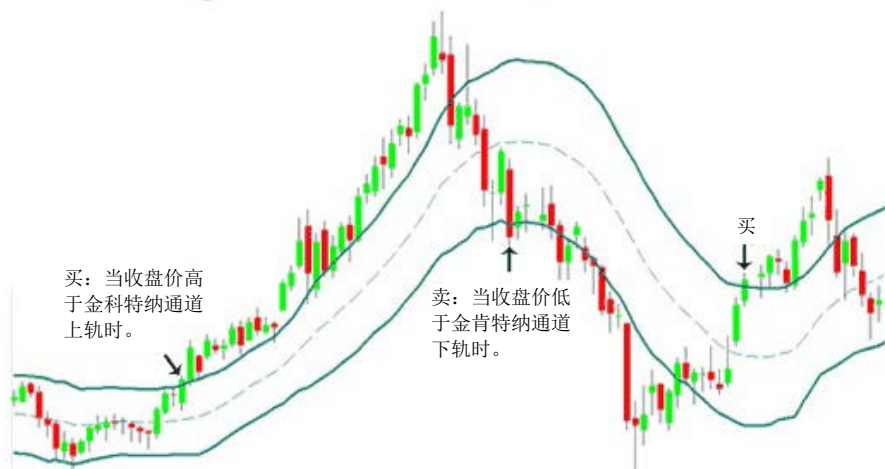


图 5-25 金肯特纳通道图示

金肯特纳通道策略三大要素如下所示。

- (1) 信号：价格突破上轨做多，突破下轨做空。
- (2) 过滤：若价格在通道内上下走动，则不进行开仓操作。
- (3) 出场：固定百分点数移动止损。

## 5.4.2 策略代码解析

### 1. 策略参数

金肯特纳通道指标初始化天数是 11 天，调用 K 线管理模块中 TA-Lib 库定义的函数可以计算出均线和 ATR 指标，进而得出金肯特纳通道的上轨和下轨，代码如下：

```
def keltner(self, n, dev, array=False):
    """金肯特纳通道"""
    mid = self.sma(n, array)
    atr = self.atr(n, array)

    up = mid + atr * dev
```

```
down = mid - atr * dev

return up, down
```

在 Keltner 函数中，入参是初始化天数（kkLeng）与通道宽度偏差（kkDev），然后把 array 设置为 True，这样就可以计算出一系列通道上轨（kkUp）和通道下轨（kkDown）。

移动止损设置为 0.8%，即在最高点回落 0.8% 或者在最低点反弹 0.8%，瞬间用停止单平仓离场。策略初始化所需要载入的天数是 10，且每次交易的合约数量为 1 手。

```
# 策略参数
kkLength = 11          # 计算通道中值的窗口数
kkDev = 1.6            # 计算通道宽度的偏差
trailingPrcent = 0.8    # 移动止损
initDays = 10          # 初始化数据所用的天数
fixedSize = 1          # 每次交易的数量
```

## 2. 新的委托方式：OCO

OCO 委托的全称是 “One-Cancels-the-Other Order”，意思是二选一委托，即在 K 线内同时发出止损买单和止损卖单：

- 若价格突破上轨，则触发止损买单同时取消止损卖单。
- 若价格突破下轨，则触发止损卖单同时取消止损买单。

这种挂单方式在国内交易所比较少见，一般多用于外汇市场。因为货币在短时间内会有很强的震荡，比较难以判断趋势，这时候 OCO 的优点就彰显出来了。当盘整震荡的行情接近结束，而要进入一个上涨或下跌的趋势时，可用 OCO 挂单捕捉趋势。一个例子是发生重大行情如非农或者利率决议公布，若不确定接下来的行情，则可用 OCO 挂单。

OCO 委托流程如下所述。

创建 3 个空的列表：buyOrderIDList、shortOrderIDList 和 ordList。



- buyOrderIDList 用于缓存止损买入单的委托，分别插入委托价格、合约手数。
- shortOrderIDList 用于缓存止损卖出单的委托，分别插入委托价格、合约手数。
- orderList 用于缓存所有发出的委托单子，用 extend() 的方法把上面两个列表添加进来。

```
#-----
# 策略变量
buyOrderIDList = []          # OCO 委托买入开仓的委托号
shortOrderIDList = []        # OCO 委托卖出开仓的委托号
orderList = []               # 保存委托代码的列表
#-----
def sendOcoOrder(self, buyPrice, shortPrice, volume):
    """
    发送 OCO 委托

    OCO(One Cancel Other)委托:
    1. 主要用于实现区间突破入场
    2. 包含两个方向相反的停止单
    3. 一个方向的停止单成交后会立即撤销另一个方向的
    """
    # 发送双边的停止单委托，并记录委托号
    self.buyOrderIDList = self.buy(buyPrice, volume, True)
    self.shortOrderIDList = self.short(shortPrice, volume, True)

    # 将委托号记录到列表中
    self.orderList.extend(self.buyOrderIDList)
    self.orderList.extend(self.shortOrderIDList)
```

### 3. OCO 发单与平仓

为保证委托的唯一性，同样要撤销之前尚未成交的委托，但这一次不采用 cancelAll() 方法，先在 orderList 列表中遍历出 orderID，然后删除，以保证清空 orderList 的目标。

调用 K 线管理模块 ArrayManager 里的 updateBar(bar) 函数把 1 分钟 K 线数据合成 5 分钟 K 线。K 线数据若不够多，则直接返回，不进行下面的操作。

先计算金肯特纳通道的上下轨。若当前无持仓，则用最新的 5 分钟 K 线数据



初始化持仓期的最高点（intraTradeHigh）和最低点（intraTradeLow），用OCO委托在上下轨道挂上突破停止单，进行开仓操作，并且将停止单委托缓存到buyOrderIDList和shortOrderIDList中。

- 若价格往上走，则触发停止买单成交，开仓做多。调用max()函数统计日高点，设置固定百分比的移动止损离场，缓存平仓委托到orderList列表中。由于OCO委托的特点，多头开仓成交后，撤销空头委托，方法在shortOrderIDList中遍历shortOrderID，然后用cancel()从列表中移除。
- 若价格往下走，则触发止损卖单成交，开仓做空。同理调用min()统计出日低点，设置离场点，发出停止买单（Stop=True），并且缓存该委托到orderList列表中。空头开仓成交后，撤销多头委托，即遍历buyOrderIDList，用cancel()移除其对应的buyOrderID。

```
def onFiveBar(self, bar):
    """收到5分钟K线"""
    # 撤销之前发出的尚未成交的委托（包括限价单和停止单）
    for orderID in self.orderList:
        self.cancelOrder(orderID)
    self.orderList = []

    # 保存K线数据
    am = self.am
    am.updateBar(bar)
    if not am.inited:
        return

    # 计算指标数值
    self.kkUp, self.kkDown = am.keltner(self.kkLength, self.kkDev)

    # 判断是否要进行交易

    # 当前无仓位，发送oco开仓委托
    if self.pos == 0:
        self.intraTradeHigh = bar.high
        self.intraTradeLow = bar.low
        self.sendOcoOrder(self.kkUp, self.kkDown, self.fixedSize)

    # 持有多头仓位
```



```

elif self.pos > 0:
    self.intraTradeHigh = max(self.intraTradeHigh, bar.high)
    self.intraTradeLow = bar.low

    l = self.sell(self.intraTradeHigh*(1-self.trailingPrcent / 100),
                  abs(self.pos), True)
    self.orderList.extend(l)

# 持有空头仓位
elif self.pos < 0:
    self.intraTradeHigh = bar.high
    self.intraTradeLow = min(self.intraTradeLow, bar.low)

    l = self.cover(self.intraTradeLow*(1+self.trailingPrcent / 100),
                  abs(self.pos), True)
    self.orderList.extend(l)

# 同步数据到数据库
self.saveSyncData()

# 发出状态更新事件
self.putEvent()

#-----
def onTrade(self, trade):
    if self.pos != 0:
        # 多头开仓成交后, 撤销空头委托
        if self.pos > 0:
            for shortOrderID in self.shortOrderIDList:
                self.cancelOrder(shortOrderID)
        # 反之同样
        elif self.pos < 0:
            for buyOrderID in self.buyOrderIDList:
                self.cancelOrder(buyOrderID)

        # 移除委托号
        for orderID in (self.buyOrderIDList + self.shortOrderIDList):
            if orderID in self.orderList:
                self.orderList.remove(orderID)

    # 发出状态更新事件
    self.putEvent()

```

### 5.4.3 策略回测

回测设置仍然是 2011—2018 年，其使用的合约数据和引擎参数没有发生变化。回测结果显示：年化收益为 18.84%，百分比最大回撤达-8.19%，夏普比率达 1.03，如图 5-26 所示。

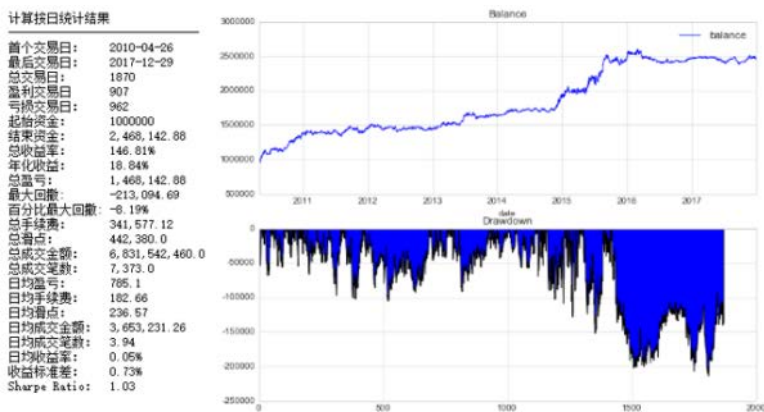


图 5-26 原始金肯特纳通道策略回测结果

### 5.4.4 滚动回测

#### 1. 参数优化设置

本次优化目标是夏普比率，需要优化的参数只有两个，分别是金肯特纳通道的回望周期和通道宽度。注意 `runParallelOptimization()` 函数会调用 CPU 的所有线程去进行并行优化，缩短优化所用的时间，缺点是电脑会变得非常卡顿。

```
# 优化配置
setting = OptimizationSetting()
setting.setOptimizeTarget('sharpeRatio')
setting.addParameter('kkLength', 8, 16, 1)
setting.addParameter('kkDev', 1, 1.8, 0.1)

# 新建一个优化任务设置对象
# 设置优化排序的目标是夏普比率
# 增加第一个优化参数 kkLength
# 增加第二个优化参数 kkDev

# 执行多进程优化
import time
start = time.time()
```

```
resultList = engine.runParallelOptimization(KkStrategy, setting)
print u'耗时: %s' %(time.time()-start)
```

## 2. 滚动回测设置

以 3 年为滚动周期，步进是半年，对 20100416 至 20180101 区间进行滚动回测及参数优化，并且收集表现最好的前 3 组优化参数。

通过对历史优化参数的统计分析来预测未来的优化参数。其选择标准如下：

- 近年参考权重大于远年。
- 若优化参数整体变化不大，则取众数。
- 若优化参数整体发生变化，则在近年区间取众数。
- 若前一年参数发生剧烈变化，则对该区间进行更为细致的滚动回测和参数优化。

如图 5-27 所示，历史优化参数整体变化不大，故取众数，即  $kkLength = 8$ ， $kkDev = 1.4$ 。同时，优化参数变化不大也说明策略稳健性强。

回测区间	最优参数		次优参数		第三优参数	
	kkLength	kkDev	kkLength	kkDev	kkLength	kkDev
20100416--20130601	12	1.6	11	1.7	11	1.5
20110101--20140101	8	1.4	8	1.2	9	1.1
20110601--20140601	9	1.7	8	1.4	8	1.6
20120101--20150101	8	1.4	8	1.3	8	1.5
20120601--20150601	8	1.4	8	1.3	8	1.5
20130101--20160101	8	1.4	8	1.5	8	1.5
20130601--20160601	8	1.4	8	1.5	8	1.3
20140101--20170101	8	1.4	8	1.3	8	1.5
20140601--20170601	8	1.4	8	1.3	8	1.5
20150101--20180101	8	1.4	8	1.3	8	1.5
2018至今	8	1.4				

图 5-27 历史优化参数

预测优化参数历史表现：年化收益为 25.28%，百分比最大回撤为-9.63%，夏普比率达 1.26，如图 5-28 所示。

对 2018 年以后的预测效果：2018 年上半年效果不理想，整体年化收益为 22.99%，百分比最大回撤达-8.24%，夏普比率为 1.02，如图 5-29 所示。

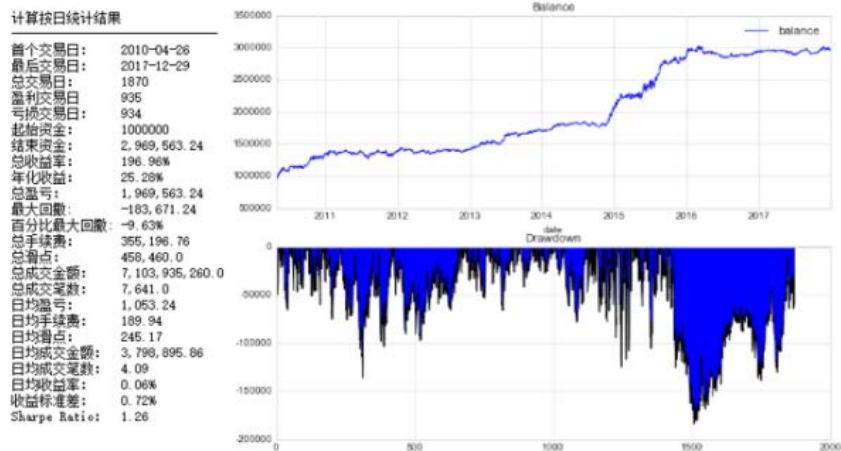


图 5-28 预测优化参数历史表现

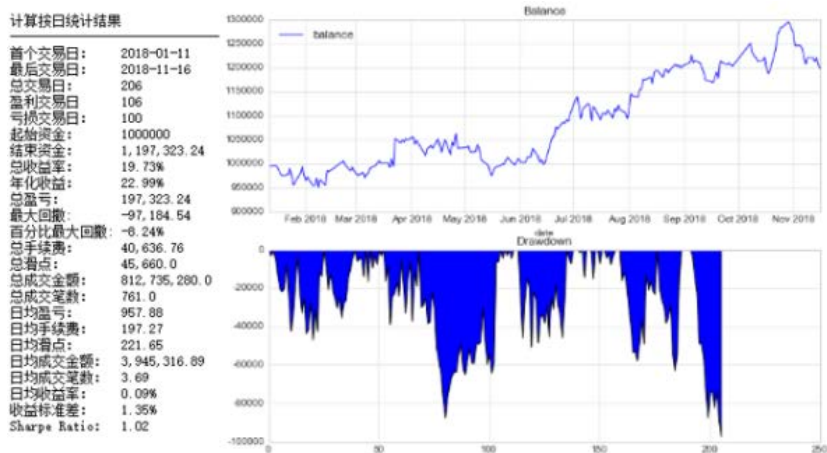


图 5-29 对 2018 年以后的预测效果

## 5.5 布林带通道策略

### 5.5.1 策略原理

布林带通道策略是一个典型的通道突破策略,即当价格突破通道上轨时做多,



当价格走低突破通道下轨时做空。轨道计算的思路也相对简单，先计算移动均线（MA），并且统计标准差 STD，设置一定的通道宽度偏差 X，则：

- 上轨 =  $MA + X \times STD$
- 下轨 =  $MA - X \times STD$

布林带通道策略与上一节介绍的金肯特纳通道策略非常类似，区别在于仅仅把 ATR 指标换成标准差。因为标准差统计的数据是基于 K 线收盘价的，它并不能捕捉到 K 线跳空高开或者跳空低开的情况，理论上更适合于一些在短期内有较小波动的品种，如商品期货。下面的策略回测会以螺纹钢期货为例，而不是之前的沪深 300 股指期货。

布林带通道策略三大要素如下所示。

- （1）信号：价格突破上轨做多，突破下轨做空。
- （2）过滤：若价格在通道内上下走动，则不进行开仓操作。
- （3）出场：固定百分点数移动止损。

在传统布林带通道策略的基础上，加入 CCI 指标做过滤，ATR 指标作为出场，策略的效果会好很多。故新版布林带通道策略三大要素如下所示。

- （1）信号：布林带通道突破开仓交易。
- （2）过滤：CCI 指标。
- （3）出场：结合 ATR 指标的移动止损。

## 5.5.2 CCI 指标

### 1. CCI 指标原理

CCI 指标又叫顺势指标，其英文全名为 Commodity Channel Index，是由美国股市分析家唐纳德·兰伯特（Donald R. Lambert）于 20 世纪 80 年代所创的，是

一种指导股市投资的中短线指标。

CCI 指标是一种超买超卖指标。所谓超买超卖指标，顾名思义，“超买”就是已经超出买方的能力，买进股票的人数超过了一定比例，因此这时候应该反向卖出股票。“超卖”则代表卖方卖股票卖过了头，卖股票的人数超过一定比例时，反而应该买进股票。这是在一般常态行情，但是如果行情是超乎寻常的强势，那些短期内暴涨暴跌的非常态行情，则超买越卖指标会突然间失去方向，行情不停地持续前进，对于这种脱序行为，CCI 指标提供了不同角度的看法，有利于投资者更好地研判行情。

## 2. CCI 计算原理

CCI 计算公式如下：

$$CCI(N \text{ 日}) = (TP - MA) \div MD \div 0.015$$

其中：

$$TP = (\text{最高价} + \text{最低价} + \text{收盘价}) \div 3$$

$$MA = \text{近 } N \text{ 日收盘价的累计之和} \div N$$

$$MD = \text{近 } N \text{ 日} (MA - \text{收盘价}) \text{ 的绝对值累计之和} \div N$$

0.015 为计算系数， $N$  为计算周期

$$CCI_{(t)} = \frac{TP_{(t)} - TPAVG_{(t)}}{0.015 \times MD_{(t)}}$$

$$TP_{(t)} = \frac{High_{(t)} + Low_{(t)} + Close_{(t)}}{3}$$

$$TPAVG_{(t)} = \frac{TP_{(t)} + TP_{(2)} + \cdots + TP_{(n)}}{n}$$



$$MD_{(t)} = \frac{ABS(TP_{(1)} - TPAVG_{(1)}) + \dots + ABS(TP_{(n)} - TPAVG_{(n)})}{n}$$

### 3. CCI 指标信号判断

- 当  $CCI > 0$  时, 判断多头趋势, K 线内用停止单做多。
- 当  $CCI < 0$  时, 判断空头趋势, K 线内用停止单做空。

### 4. 策略代码

CCI 指标过滤与布林带通道交易信号相结合, 主要用于开仓操作。若  $CCI > 0$ , 显示上升趋势, 则在布林带通道上轨挂上买入停止单。若  $CCI < 0$ , 显示下降趋势, 则在布林带通道下轨挂上卖出停止单。

```
def onXminBar(self, bar):
    .....

    # 当前无仓位, 发送开仓委托
    if self.pos == 0:
        self.intraTradeHigh = bar.high
        self.intraTradeLow = bar.low

        if self.cciValue > 0:
            self.buy(self.bollUp, self.fixedSize, True)

        elif self.cciValue < 0:
            self.short(self.bollDown, self.fixedSize, True)

    .....
```

## 5.5.3 ATR 指标

### 1. ATR 指标的作用

ATR 指标主要用来衡量市场波动的强烈程度, 即用于显示市场波动率的指标。它主要用来衡量价格的波动, 并不能直接反映价格走向及其趋势稳定性。当 ATR 增大时, 说明市场波动率高, 市场上下跳动的幅度很大, 止损位置可以相对设置得远一些; 当 ATR 减小时, 说明行情波动不大, 这时候需要把止损位置设置得近



一点，以锁定利润。所以，用ATR指标比固定百分点位的移动止损在行情不断变化的市场效果会好一些。

## 2. 策略代码

开仓之后用到ATR指标移动止损。当持有多仓时，统计日高点，设置移动止损为日高点减去一定点数的ATR指标，通过卖出停止单离场；同理，当持有空仓时，设置移动止损为日低点加上一定点数的ATR指标，通过买入停止单离场。

```
# 持有多头仓位
elif self.pos > 0:
    self.intraTradeHigh = max(self.intraTradeHigh, bar.high)
    self.intraTradeLow = bar.low
    self.longStop = self.intraTradeHigh - self.atrValue * self.slMultiplier

    self.sell(self.longStop, abs(self.pos), True)

# 持有空头仓位
elif self.pos < 0:
    self.intraTradeHigh = bar.high
    self.intraTradeLow = min(self.intraTradeLow, bar.low)
    self.shortStop = self.intraTradeLow + self.atrValue * self.slMultiplier

    self.cover(self.shortStop, abs(self.pos), True)
```

### 5.5.4 策略回测

布林带通道策略是专门对螺纹钢期货指数进行优化的，策略回测结果：年化收益为30.56%，百分比最大回撤为-16.32%，夏普比率为1.11，如图5-30所示。



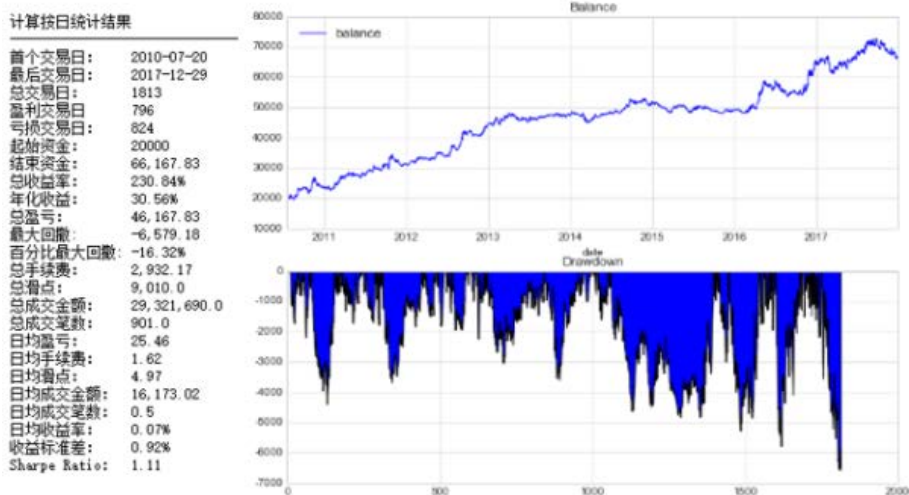


图 5-30 原始布林带通道回测结果

## 5.5.5 滚动回测

### 1. 参数优化设置

优化目标同样是夏普比率，优化的参数分为 3 个维度，分别计算布林带通道所需要的回望窗口数、通道的宽度，以及 CCI 指标的回望窗口数。

```
# 优化配置
setting = OptimizationSetting()
setting.setOptimizeTarget('sharpeRatio')
setting.addParameter('bollWindow', 18, 30, 2)
setting.addParameter('bollDev', 3.2, 4, 0.2)
setting.addParameter('cciWindow', 12, 16, 1)

# 新建一个优化任务设置对象
# 设置优化排序的目标是夏普比率

# 执行多进程优化
import time
start = time.time()
resultList = engine.runParallelOptimization(BollChannelStrategy, setting)
print u'耗时: %s' %(time.time()-start)
```

## 2. 滚动回测设置

以 3 年为滚动周期，步进是半年，对 20100416 至 20180101 区间进行滚动回测及参数优化，并且收集表现最好的前 3 组优化参数。

如图 5-31 所示，最后一个回测区间，即 20150101 至 20180101，其优化参数发生剧烈变化，上一个回测区间 20140601 至 20170601 有非常大的不同，说明行情发生巨大变化。

回测区间	最优参数			次优参数			第三优参数		
	bollDev	bollWindow	cciWindow	bollDev	bollWindow	cciWindow	bollDev	bollWindow	cciWindow
20100416--20130601	3.8	30	15	3.4	18	13	3.4	18	12
20110101--20140101	3.4	18	13	3.4	18	12	3.2	18	12
20110601--20140601	3.8	30	15	3.2	22	16	3.2	22	12
20120101--20150101	3.2	22	16	3.2	22	12	3.2	22	15
20120601--20150601	3.2	22	16	3.2	22	12	3.2	22	13
20130101--20160101	3.2	24	12	3.2	24	13	3.2	24	14
20130601--20160601	3.2	24	12	3.2	24	13	3.2	24	15
20140101--20170101	3.2	24	12	3.2	24	13	3.2	24	15
20140601--20170601	3.2	30	14	3.2	30	15	3.2	30	16
20150101--20180101	3.8	18	14	3.8	18	13	3.8	18	16

图 5-31 历史优化参数

故对以上两个区间进行更加细致的回测：以 1 年为滚动周期，步进是半年。回测效果如图 5-32 所示。对近年优化参数选取众数，得 bollDev = 3.2，bollWindow = 18，cciWindow = 16。

回测区间	最优参数			次优参数			第三优参数		
	bollDev	bollWindow	cciWindow	bollDev	bollWindow	cciWindow	bollDev	bollWindow	cciWindow
20140601--20150601	3.6	28	12	3.6	28	13	3.6	28	14
20150101--20160101	3.2	24	13	3.2	24	12	3.2	24	14
20150601--20160601	3.8	30	13	3.2	30	12	3.2	30	13
20160101--20170101	3.8	30	12	3.8	30	13	3.8	30	14
20160601--20170601	3.8	18	14	3.8	18	16	3.8	18	12
20170101--20180101	3.8	18	16	3.8	18	12	3.6	18	16
2018至今	3.8	18	16						

图 5-32 更细致的回测

预测优化参数历史表现：年化收益为 25.65%，百分比最大回撤为-21.04%，夏普比率达 0.91，如图 5-33 所示。

对 2018 年以后的预测效果：年化收益为 23.35%，百分比最大回撤为-17.3%，夏普比率为 0.65，优化参数预测效果不佳，如图 5-34 所示。



计算按日统计结果

首个交易日: 2010-07-20  
 最后交易日: 2017-12-29  
 总交易日: 1813  
 盈利交易日: 714  
 亏损交易日: 771  
 起始资金: 20000  
 结束资金: 58,760.09  
 总收益率: 193.8%  
 年化收益: 25.65%  
 总盈亏: 38,760.09  
 最大回撤: -5,985.7  
 百分比最大回撤: -21.04%  
 总手续费: 2,669.91  
 总滑点: 8,190.0  
 总成交金额: 26,699,080.0  
 总成交笔数: 819.0  
 日均盈亏: 21.38  
 日均手续费: 1.47  
 日均滑点: 4.52  
 日均成交金额: 14,726.46  
 日均成交笔数: 0.45  
 日均收益率: 0.06%  
 收益标准差: 1.01%  
 Sharpe Ratio: 0.91

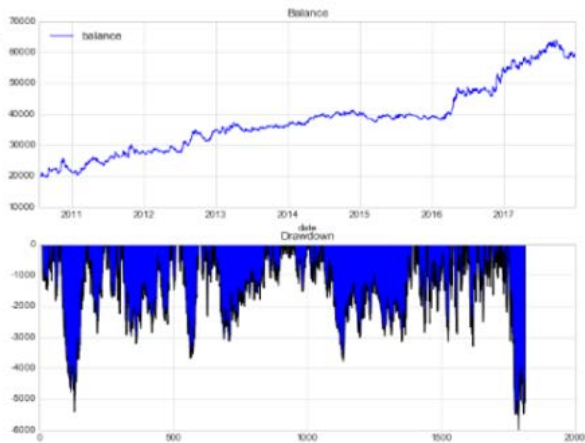


图 5-33 预测优化参数历史表现

计算按日统计结果

首个交易日: 2018-01-11  
 最后交易日: 2018-11-05  
 总交易日: 197  
 盈利交易日: 85  
 亏损交易日: 105  
 起始资金: 20000  
 结束资金: 23,833.52  
 总收益率: 19.17%  
 年化收益: 23.35%  
 总盈亏: 3,833.52  
 最大回撤: -4,248.92  
 百分比最大回撤: -17.3%  
 总手续费: 496.48  
 总滑点: 1,290.0  
 总成交金额: 4,964,830.0  
 总成交笔数: 129.0  
 日均盈亏: 19.46  
 日均手续费: 2.52  
 日均滑点: 6.55  
 日均成交金额: 25,202.18  
 日均成交笔数: 0.65  
 日均收益率: 0.08%  
 收益标准差: 1.98%  
 Sharpe Ratio: 0.65

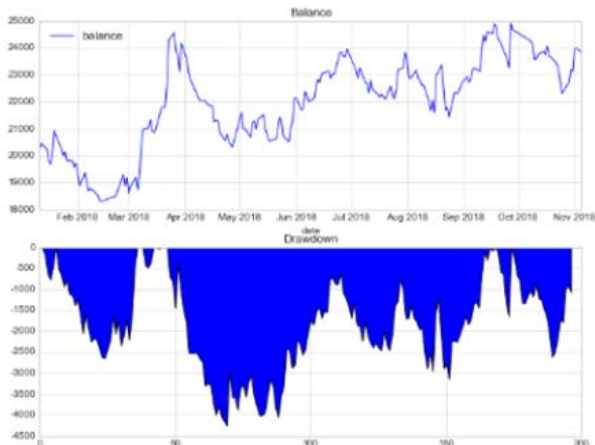


图 5-34 对 2018 年以后的预测效果

## 5.6 跨时间周期策略

之前介绍的 CTA 经典策略的过滤、信号和出场都基于同一时间周期，但是在实际应用中，更长的 K 线周期能够过滤掉噪声，易于判断趋势；更短的 K 线周期能够充分利用短期的波动，易于找到合适的进出场位置。跨时间周期策略结合两

者的特点，其适用范围会比单一周期策略更广。

## 5.6.1 策略原理

该策略结合不同时间周期的优势。长时间周期用于判断趋势，如在 15 分钟 K 线周期上，若快均线上穿慢均线形成金叉，则判断为上涨趋势；反之，若快均线下穿慢均线形成死叉，则判断为下跌趋势。短时间周期用于具体入场信号生成，如基于 5 分钟 K 线周期：在上涨趋势中，若  $RSI >$  规定上限，开仓做多；在下跌趋势中，若  $RSI <$  规定下限，开仓做空。当判断到上涨趋势结束或者  $RSI < 50$ ，多头平仓；当判断到下跌趋势结束或者  $RSI > 50$ ，空头平仓。

总结一下，跨时间周期策略三大因素如下所示。

- (1) 过滤：15 分钟 K 线周期双均线。
- (2) 信号：5 分钟 K 线周期 RSI 信号。
- (3) 出场：不同时间周期任何一个条件不满足，则平仓离场。

## 5.6.2 策略代码解析

### 1. 策略参数

快均线窗口和慢均线窗口用于计算移动平均值；RSI 窗口用于计算 RSI 指标，然后与规定的上限/下限对比。规定上限 =  $50 + \text{RSI 信号阈值}$ ，规定下限 =  $50 - \text{RSI 阈值}$ 。

初始天数为 10 天，初始化完成后开始交易，每次交易的数量为 1 手。

```
# 策略参数
rsiSignal = 20      # RSI 信号阈值
rsiWindow = 14      # RSI 窗口
fastWindow = 5      # 快均线窗口
slowWindow = 20     # 慢均线窗口
```



```
initDays = 10          # 初始化数据所用的天数
fixedSize = 1          # 每次交易的数量
```

## 2. 初始化策略

在 `init()` 函数下先计算 `RSI` 指标规定的上限和下限。由于要用到两个时间周期，故需要调用两次 `K` 线合成模块和 `K` 线管理模块，用于合成不同时间周期的 `K` 线，以及合成基于各时间周期的计算指标。

```
def __init__(self, ctaEngine, setting):
    """Constructor"""
    super(MultiTimeframeStrategy, self).__init__(ctaEngine, setting)

    self.rsiLong = 50 + self.rsiSignal
    self.rsiShort = 50 - self.rsiSignal

    # 创建K线合成器对象
    self.bg5 = BarGenerator(self.onBar, 5, self.on5MinBar)
    self.am5 = ArrayManager()

    self.bg15 = BarGenerator(self.onBar, 15, self.on15MinBar)
    self.am15 = ArrayManager()
```

## 3. 合成不同时间周期的 K 线

在 `onTick` 函数下，调用 `K` 线生成模块的 `updateTick` 可以基于 `Tick` 级别数据合成 1 分钟 `K` 线。在 `onBar` 函数下，`updateBar` 则基于 1 分钟 `K` 线分别合成 15 分钟周期和 5 分钟周期的 `K` 线。

```
def onTick(self, tick):
    """收到行情 Tick 推送（必须由用户继承实现）"""
    # 只需要在一个 BarGenerator 中合成 1 分钟 K 线
    self.bg5.updateTick(tick)

    #-----
    def onBar(self, bar):
        """收到 Bar 推送（必须由用户继承实现）"""
        # 基于 15 分钟判断趋势过滤，因此先更新
        self.bg15.updateBar(bar)
```

```
# 基于 5 分钟判断
self.bg5.updateBar(bar)
```

#### 4. 15 分钟判断

若没有插入足够数据，初始化状态为 False，直接返回；反之，调用 TA-Lib 定义的 sma 函数分别计算快均线与慢均线。若快均线上穿慢均线，则判断为上涨趋势（self.meTrend = 1）；若快均线下穿慢均线，判断为下跌趋势（self.maTrend = -1）。

```
def on15MinBar(self, bar):
    """15 分钟 K 线推送"""
    self.am15.updateBar(bar)

    if not self.am15.inited:
        return

    # 计算均线并判断趋势
    self.fastMa = self.am15.sma(self.fastWindow)
    self.slowMa = self.am15.sma(self.slowWindow)

    if self.fastMa > self.slowMa:
        self.maTrend = 1
    else:
        self.maTrend = -1
```

#### 5. 5 分钟判断

用 cancel() 清空未成交委托，若初始化完成 5 分钟周期和 15 分钟周期的 K 线数据，则调用 TA-Lib 定义的 rsi 函数计算 RSI 指标。

```
def on5MinBar(self, bar):
    """5 分钟 K 线"""
    self.cancelAll()

    # 保存 K 线数据
    self.am5.updateBar(bar)
    if not self.am5.inited:
        return
```



```
# 如果 15 分钟数据尚未初始化完毕, 则直接返回
if not self.maTrend:
    return

# 计算指标数值
self.rsiValue = self.am5.rsi(self.rsiWindow)
```

- **当前无持仓**, 若基于 15 分钟周期 K 线判断为上涨趋势并且 RSI 值大于规定上限, 则用高于收盘价 5 个点的限价单保证多仓成交; 若基于 15 分钟周期 K 线判断为下跌趋势并且 RSI 值小于规定下限, 则用低于收盘价 5 个点的限价单保证空仓成交。
- **当前持多仓**, 若判断为下跌趋势或者  $RSI < 50$ , 则用低于收盘价 5 个点的限价单对多头平仓。
- **当前持空仓**, 若判断为上涨趋势或者  $RSI > 50$ , 则用高于收盘价 5 个点的限价单对空头平仓。

```
# 判断是否要进行交易

# 当前无仓位
if self.pos == 0:
    if self.maTrend > 0 and self.rsiValue >= self.rsiLong:
        self.buy(bar.close+5, self.fixedSize)

    elif self.maTrend < 0 and self.rsiValue <= self.rsiShort:
        self.short(bar.close-5, self.fixedSize)

# 持有多头仓位
elif self.pos > 0:
    if self.maTrend < 0 or self.rsiValue < 50:
        self.sell(bar.close - 5, abs(self.pos))

# 持有空头仓位
elif self.pos < 0:
    if self.maTrend > 0 or self.rsiValue > 50:
        self.cover(bar.close + 5, abs(self.pos))

# 发出状态更新事件
self.putEvent()
```



### 5.6.3 策略回测

本次策略回测使用的数据是从 Mc8s 下载的天然橡胶期货指数数据( ru0000 ), 回测时间是 2014—2018 年。天然橡胶合约规模是 10 吨每手, 最小价格变动为 5 元/吨, 手续费率为万分之 0.45, 滑点等于最小价格变动, 起始资金是 20 万元人民币。

```
# 创建回测引擎对象
engine = BacktestingEngine()
# 设置回测使用的数据
engine.setBacktestingMode(engine.BAR_MODE) # 设置引擎的回测模式为 K 线
engine.setDatabase(MINUTE_DB_NAME, 'ru0000') # 设置使用的历史数据库
engine.setStartDate('20140101') # 设置回测用的数据起始日期
engine.setEndDate('20180101') # 设置回测用的数据结束日期
# 配置回测引擎参数
engine.setSlippage(5) # 设置滑点
engine.setRate(0.45/10000) # 设置手续费为万分之 0.45
engine.setSize(10) # 设置股指合约大小
engine.setPriceTick(5) # 设置股指最小价格变动
engine.setCapital(200000) # 设置回测本金
```

策略回测结果: 年化收益为 11.82%, 百分比最大回撤为-7.77%, 夏普比率达 0.86, 如图 5-35 所示。

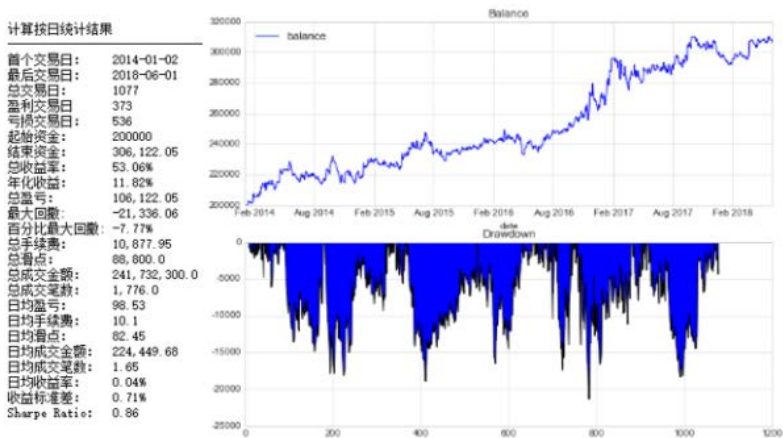


图 5-35 原始跨时间周期策略回测结果



## 5.6.4 滚动回测

### 1. 参数优化设置

参数优化目标是夏普比率，参数分为 3 个维度，分别计算快/慢均线的回望窗口数及 RSI 值的回望窗口数。

```
# 优化配置
setting = OptimizationSetting()                    # 新建一个优化任务设置对象
setting.setOptimizeTarget('sharpeRatio')          # 设置优化排序的目标是夏普比率
setting.addParameter('fastWindow', 9, 14, 1)
setting.addParameter('slowWindow', 14, 22, 2)
setting.addParameter('rsiWindow', 12, 16, 2)
# 执行多进程优化
import time
start = time.time()
resultList = engine.runParallelOptimization(MultiTimeframeStrategy, setting)
print u'耗时: %s' %(time.time()-start)
```

### 2. 滚动回测

以 3 年为滚动周期，步进是半年，对 20100416 至 20180101 区间进行滚动回测，以及参数优化，并且收集表现最好的前 3 组优化参数。

如图 5-36 所示，由于历史优化参数整体变化不大，故取众数，rsiWindow = 14，fastWindow = 12，slowWindow = 20。

回测区间	最优参数			次优参数			第三优参数		
	rsiWindow	fastWindow	slowWindow	rsiWindow	fastWindow	slowWindow	rsiWindow	fastWindow	slowWindow
20140101--20150101	14	12	20	14	10	24	14	12	22
20140601--20150601	14	12	18	14	12	20	14	6	24
20150101--20160101	14	12	20	16	12	20	14	12	18
20150601--20160601	14	12	20	14	10	22	14	8	24
20160101--20170101	16	12	18	16	12	20	16	10	20
20160601--20170601	16	10	20	16	12	18	16	10	22
20170101--20180101	18	10	22	18	10	22	18	12	24
2018至今	14	12	20						

图 5-36 历史优化参数

预测优化参数历史表现：年化收益为 15.84%，百分比最大回撤为-8.34%，夏普比率达 1.27，如图 5-37 所示。

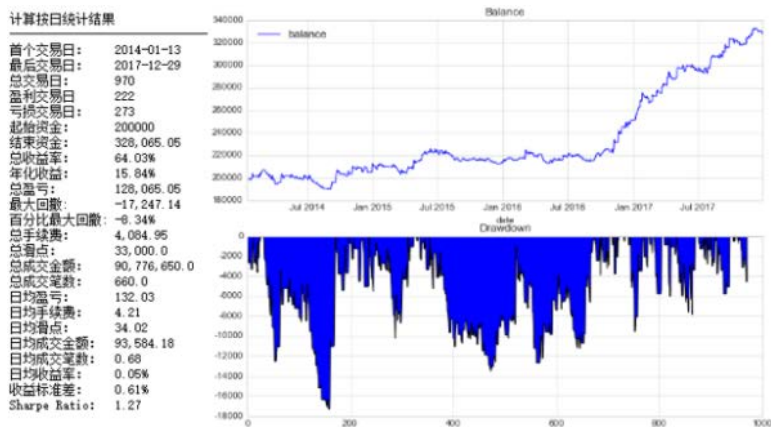


图 5-37 预测优化参数历史表现

对 2018 年以后的预测效果: 年化收益为 15.11%, 百分比最大回撤达-3.16%, 夏普比率为 1.72, 优化参数预测效果理想, 如图 5-38 所示。

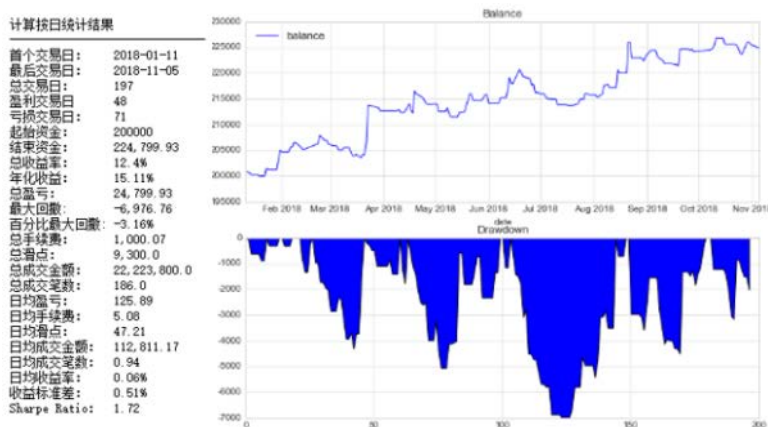


图 5-38 对 2018 年以后的预测效果

## 5.7 多信号组合策略

简单的 CTA 策略会把信号生成和交易管理放在同一个类上, 基于固定模式的逻辑判断结构, 虽易于策略回测和参数优化, 但不利于策略的扩展及提高复杂度。

vn.py 提供的解决方案是把信号生成和交易管理两部分独立开来，信号生成模块是基于特定周期的 K 线数据，计算技术指标，通过其判断生成交易信号；而交易管理仅仅专注于下单算法领域，用于降低冲击成本，更快、更好地获得成交价格。

## 5.7.1 策略原理

多信号组合策略的工作原理类似于股票的多因子 Alpha 策略的打分法，在接收行情推送时，各因子产生的交易信号可以看成分数，分数汇总后才进行下单交易。分数是交易管理的基准，如 1 分 = 1 手多单，0 分 = 不交易，-1 分 = 1 手空单。

当然，这是多信号组合策略的最简版，进阶版可以是：给予因子各自的权重，分数汇总后高于一定阈值做多，低于一定阈值做空。更复杂的是加入机器学习模型，从巨型因子中筛选出因子池。

## 5.7.2 信号生成部分

### 1. CTA 策略信号

CtaSignal 属于 CtaTemlpe 的简化版，负责纯粹的信号生成，不参与具体交易管理。CtaSignal 仅包含 5 个函数，分别是\_\_init\_\_、onTick、onBar、setSignalPos 和 getSignalPos。

应该注意的是，setSignalPos 通过内部逻辑运算来设置信号仓位，getSignalPos 用于把计算好的信号仓位供外部调用。

```
class CtaSignal(object):
    """
    CTA 策略信号，负责纯粹的信号生成（目标仓位），不参与具体交易管理
    """
    .....
    def setSignalPos(self, pos):
        """设置信号仓位"""
        self.signalPos = pos
```

```
#-----  
def getSignalPos(self):  
    """获取信号仓位"""  
    return self.signalPos
```

## 2. RSI 信号

RSI 信号继承父类 CtaSignal 模板，在\_\_init\_\_下设置策略参数，如计算 RSI 指标的滚动窗口数 = 14，RSI 指标开仓阈值 = 20，规定上限 = 50 + 开仓阈值，规定下限 = 50 - 开仓阈值。本地化 K 线生成模块和 K 线管理模块。

onTick 函数：调用 K 线生成模块的 updateTick 函数，把 Tick 数据合成 1 分钟 K 线数据。

onBar 函数：若 K 线管理模块未完成初始化，则信号仓位设为 0（信号仓位仅仅是数字，不是持仓命令）；反之，若初始化完成，则基于 1 分钟 K 线数据合成 RSI 指标。

- 若 RSI 指标大于或等于规定上限，则将信号仓位改成数字 1。
- 若 RSI 指标小于或等于规定下限，则将信号仓位改成数字-1。
- 若 RSI 指标在上下限之间，则信号仓位仍然是 0。

```
class RsiSignal(CtaSignal):  
    """RSI 信号"""  
  
    #-----  
    def __init__(self):  
        """Constructor"""  
        super(RsiSignal, self).__init__()  
  
        self.rsiWindow = 14  
        self.rsiLevel = 20  
        self.rsiLong = 50 + self.rsiLevel  
        self.rsiShort = 50 - self.rsiLevel  
  
        self.bg = BarGenerator(self.onBar)  
        self.am = ArrayManager()
```



```

#-----
def onTick(self, tick):
    """Tick 更新"""
    self.bg.updateTick(tick)

#-----
def onBar(self, bar):
    """K 线更新"""
    self.am.updateBar(bar)

    if not self.am.inited:
        self.setSignalPos(0)

    rsiValue = self.am.rsi(self.rsiWindow)

    if rsiValue >= self.rsiLong:
        self.setSignalPos(1)
    elif rsiValue <= self.rsiShort:
        self.setSignalPos(-1)
    else:
        self.setSignalPos(0)

```

### 3. CCI 信号

CCI 信号继承父类 CtaSignal 模板，在\_\_init\_\_下设置策略参数，如计算 CCI 指标的滚动窗口数 = 30，CCI 指标开仓阈值 = 10，规定上限 = 开仓阈值，规定下限 = -开仓阈值。本地化 K 线生成模块和 K 线管理模块。

onTick 函数：调用 K 线生成模块的 updateTick 函数，把 Tick 数据合成 1 分钟 K 线数据。

onBar 函数：若 K 线管理模块未完成初始化，则将信号仓位设为 0；反之，若初始化完成，则基于 1 分钟 K 线数据合成 CCI 指标。

- 若 CCI 指标大于或等于规定上限，则将信号仓位改成数字 1。
- 若 CCI 指标小于或等于规定下限，则将信号仓位改成数字-1。
- 若 CCI 指标在上下限之间，则信号仓位仍然是 0。

```
class CciSignal(CtaSignal):
    """CCI 信号"""

    #-----
    def __init__(self):
        """Constructor"""
        super(CciSignal, self).__init__()

        self.cciWindow = 30
        self.cciLevel = 10
        self.cciLong = self.cciLevel
        self.cciShort = -self.cciLevel

        self.bg = BarGenerator(self.onBar)
        self.am = ArrayManager()

    #-----
    def onTick(self, tick):
        """Tick 更新"""
        self.bg.updateTick(tick)

    #-----
    def onBar(self, bar):
        """K 线更新"""
        self.am.updateBar(bar)

        if not self.am.inited:
            self.setSignalPos(0)

        cciValue = self.am.cci(self.cciWindow)

        if cciValue >= self.cciLong:
            self.setSignalPos(1)
        elif cciValue <= self.cciShort:
            self.setSignalPos(-1)
        else:
            self.setSignalPos(0)
```

#### 4. 双均线信号

双均线信号继承父类 CtaSignal 模板，在\_\_init\_\_下设置策略参数，如计算快均线指标的滚动窗口数=5，慢均线指标的滚动窗口数=20。本地化 K 线生成模块



和 K 线管理模块。

**onTick 函数：**调用 K 线生成模块的 **updateTick** 函数，把 Tick 数据合成 1 分钟 K 线数据。

**onBar 函数：**调用 K 线生成模块的 **updateBar** 函数，把 1 分钟 K 线数据合成 5 分钟 K 线数据。

**onFiveBar 函数：**若 K 线管理模块未完成初始化，则将信号仓位设为 0；反之，若初始化完成，则基于 5 分钟 K 线数据合成快均线和慢均线。

- 若快均线上穿慢均线，则判断为金叉，将信号仓位改成数字 1。
- 若快均线下穿慢均线，则判断为死叉，将信号仓位改成数字-1。
- 若不符合以上二者，信号仓位仍然是 0。

```
class MaSignal(CtaSignal):
    """双均线信号"""

    #-----
    def __init__(self):
        """Constructor"""
        super(MaSignal, self).__init__()

        self.fastWindow = 5
        self.slowWindow = 20

        self.bg = BarGenerator(self.onBar, 5, self.onFiveBar)
        self.am = ArrayManager()

    #-----
    def onTick(self, tick):
        """Tick 更新"""
        self.bg.updateTick(tick)

    #-----
    def onBar(self, bar):
        """K 线更新"""
        self.bg.updateBar(bar)
```



```
#-----  
def onFiveBar(self, bar):  
    """5 分钟 K 线更新"""  
    self.am.updateBar(bar)  
  
    if not self.am.inited:  
        self.setSignalPos(0)  
  
    fastMa = self.am.sma(self.fastWindow)  
    slowMa = self.am.sma(self.slowWindow)  
  
    if fastMa > slowMa:  
        self.setSignalPos(1)  
    elif fastMa < slowMa:  
        self.setSignalPos(-1)  
    else:  
        self.setSignalPos(0)
```

### 5.7.3 交易管理部分

#### 1. 目标持仓算法

目标持仓算法不关心具体信号生成，只专注下单算法模块，允许直接通过修改目标持仓来实现交易。开发策略时，无须调用 buy/sell/cover/short 这些具体的委托指令，只需在策略逻辑运行完成后调用 setTargetPos 设置目标持仓即可，底层算法会自动完成相关交易。

下面代码中的 trade 函数工作原理：先撤销未成交委托，检查目标持仓与实际持仓的一致性，若二者一致，则不进行操作；反之，分两种情况操作。

##### (1) Bar 模式

委托价格：回测环境一般只用到 1 分钟 K 线数据，故委托价是基于分钟 K 线数据。若目标持仓大于实际持仓，则发出多单委托，委托价=最新分钟 K 线数据收盘价+超价；若目标持仓小于实际持仓，则发出空单委托，委托价=最新分钟 K 线收盘价-超价。



委托方式：若目标持仓大于实际持仓，则买入开仓，开仓价格为计算好的多单委托价，仓位为目标持仓与实际持仓之差的绝对值。反之，卖出开仓，开仓价格为计算好的空单委托价，仓位为目标持仓与实际持仓之差的绝对值。发出委托后缓存委托单 1 至委托列表 `orderList` 中。

## (2) Tick 模式

委托价格：实盘中只能收到 Tick 级别数据推送，故委托价是基于 Tick 数据的。若目标持仓大于实际持仓，则发出多单委托，委托价=最新 Tick 数据的买 1 价+超价，多单委托价不能高于涨停价；若目标持仓小于实际持仓，则发出空单委托，委托价=最新 Tick 数据的卖 1 价+超价，空单委托价不能低于跌停价。

委托方式：确保清空委托列表 `orderList` 后，若目标持仓大于实际持仓，则判断为做多状态。此时当前持空仓，若买入量小于空头持仓量，则直接平空买入量；反之，若买入量大于空头持仓量，则先平掉空头持仓量。若当前无空仓，则直接开仓买入。

若目标持仓小于实际持仓，则判断为做空状态。此时当前持多仓，若卖出量小于多头持仓量，则直接平多卖出量；反之，若卖出量大于多头持仓量，则先平掉多头持仓量。若当前无多仓，则直接开仓卖出。以上发出的委托单 1 需要缓存至委托列表 `orderList` 中。

```
class TargetPosTemplate(CtaTemplate):
    .....
    # 目标持仓模板的基本变量
    tickAdd = 1          # 委托时相对基准价格的超价
    lastTick = None      # 最新 Tick 数据
    lastBar = None       # 最新 Bar 数据
    targetPos = EMPTY_INT # 目标持仓
    orderList = []       # 委托号列表
    .....
    def setTargetPos(self, targetPos):
        """设置目标仓位"""
        self.targetPos = targetPos
```

```
self.trade()

#-----
def trade(self):
    """执行交易"""
    # 先撤销之前的委托
    self.cancelAll()

    # 如果目标仓位和实际仓位一致，则不进行任何操作
    posChange = self.targetPos - self.pos
    if not posChange:
        return

    # 确定委托基准价格，有 Tick 数据时优先使用，否则使用 Bar 数据
    longPrice = 0
    shortPrice = 0

    if self.lastTick:
        if posChange > 0:
            longPrice = self.lastTick.askPrice1 + self.tickAdd
            if self.lastTick.upperLimit:
                longPrice = min(longPrice, self.lastTick.upperLimit)
# 涨停价检查
        else:
            shortPrice = self.lastTick.bidPrice1 - self.tickAdd
            if self.lastTick.lowerLimit:
                shortPrice = max(shortPrice, self.lastTick.lowerLimit)
# 跌停价检查
    else:
        if posChange > 0:
            longPrice = self.lastBar.close + self.tickAdd
        else:
            shortPrice = self.lastBar.close - self.tickAdd

    # 回测模式下，采用合并平仓和反向开仓委托的方式
    if self.getEngineType() == ENGINETYPE_BACKTESTING:
        if posChange > 0:
            l = self.buy(longPrice, abs(posChange))
        else:
            l = self.short(shortPrice, abs(posChange))
        self.orderList.extend(l)
```



```

# 实盘模式下, 首先确保之前的委托都已经结束 (全成、撤销)
# 然后先发平仓委托, 等待成交后, 再发送新的开仓委托
else:
    # 检查之前委托都已结束
    if self.orderList:
        return

    # 买入
    if posChange > 0:
        # 若当前有空头持仓
        if self.pos < 0:
            # 若买入量小于空头持仓, 则直接平空买入量
            if posChange < abs(self.pos):
                l = self.cover(longPrice, posChange)
            # 否则先平所有的空头仓位
            else:
                l = self.cover(longPrice, abs(self.pos))
            # 若没有空头持仓, 则执行开仓操作
            else:
                l = self.buy(longPrice, abs(posChange))
        # 卖出和以上相反
    else:
        if self.pos > 0:
            if abs(posChange) < self.pos:
                l = self.sell(shortPrice, abs(posChange))
            else:
                l = self.sell(shortPrice, abs(self.pos))
            else:
                l = self.short(shortPrice, abs(posChange))
    self.orderList.extend(l)

```

## 2. 多信号组合策略

`__init__` 函数: 继承父类目标持仓算法模板, 实例化信号生成类 (CCI 信号、RSI 信号、双均线信号), 初始化信号字典。

`onTick` 函数: 调用各信号合成的 1 分钟 K 线, 计算目标持仓 (用于实盘环境)。

`onBar` 函数: 调用各信号合成的 X 分钟 K 线和技术指标, 计算目标持仓 (用于回测环境)。



calculateTargetPos 函数：把各因子产生的信号仓位缓存到信号字典，遍历信号字典计算最终的目标持仓，最后调用 setTargetPos 函数，传到底层的目标持仓算法进行具体委托交易。

```
class MultiSignalStrategy(TargetPosTemplate):
    .....
    #-----
    def __init__(self, ctaEngine, setting):
        """Constructor"""
        super(MultiSignalStrategy, self).__init__(ctaEngine, setting)

        self.rsiSignal = RsiSignal()
        self.cciSignal = CciSignal()
        self.maSignal = MaSignal()

        self.signalPos = {
            "rsi": 0,
            "cci": 0,
            "ma": 0
        }
    .....

    #-----
    def onTick(self, tick):
        """收到行情 Tick 推送（必须由用户继承实现）"""
        super(MultiSignalStrategy, self).onTick(tick)

        self.rsiSignal.onTick(tick)
        self.cciSignal.onTick(tick)
        self.maSignal.onTick(tick)

        self.calculateTargetPos()

    #-----
    def onBar(self, bar):
        """收到 Bar 推送（必须由用户继承实现）"""
        super(MultiSignalStrategy, self).onBar(bar)

        self.rsiSignal.onBar(bar)
        self.cciSignal.onBar(bar)
```



```

        self.maSignal.onBar(bar)

        self.calculateTargetPos()

#-----
def calculateTargetPos(self):
    """计算目标仓位"""
    self.signalPos['rsi'] = self.rsiSignal.getSignalPos()
    self.signalPos['cci'] = self.cciSignal.getSignalPos()
    self.signalPos['ma'] = self.maSignal.getSignalPos()

    targetPos = 0
    for v in self.signalPos.values():
        targetPos += v

    self.setTargetPos(targetPos)

.....

```

### 5.7.4 多信号策略的重构

原版多信号策略的优点是把信号生成和交易管理分离，为策略的研究提供新的思路。但是在信号生成部分已经把策略参数写死，因此参数的优化成为一个难题。

该问题的解决方案是对信号部分，如 `CciSignal` 类，重新封装。封装后的策略参数可由其他类提供，如 `MultiSignalStrategy` 类。信号部分重构示例如下：

(1) `__init__` 函数删除需要初始化的策略参数。

(2) `onBar` 函数增加两个传入的参数，如计算 CCI 指标的回望窗口数 `cciWindow` 和开仓阈值 `cciLevel`。

```

class CciSignal(CtaSignal):
    """CCI 信号"""

#-----
def __init__(self, cciWindow, cciLevel):
    """Constructor"""

```

```

super(CciSignal, self).__init__()

self.bg = BarGenerator(self.onBar)
self.am = ArrayManager()

self.cciWindow = cciWindow
self.cciLevel = cciLevel
self.cciLong = 50+self.cciLevel
self.cciShort = 50-self.cciLevel

#-----
def onTick(self, tick):
    """Tick 更新"""
    self.bg.updateTick(tick)

#-----
def onBar(self, bar):
    """K 线更新"""
    self.am.updateBar(bar)

    if not self.am.inited:
        self.setSignalPos(0)

    cciValue = self.am.cci(self.cciWindow)

    if cciValue >= self.cciLong:
        self.setSignalPos(1)
    elif cciValue <= self.cciShort:
        self.setSignalPos(-1)
    else:
        self.setSignalPos(0)

```

在交易管理部分，如 `MultiSignalStrategy` 类，则需要增加信号部分的策略参数，并且把该参数名称添加到参数列表 `paraList` 中，以便优化。

然后在 `onBar` 函数中传入定义好的策略参数即可，其示例代码如下：

```

.....
# 策略参数
rsiWindow= 14
rsiLevel = 20

```



```

cciWindow = 30
cciLevel = 10

fastWindow = 5
slowWindow = 20

initDays = 10          # 初始化数据所用的天数
fixedSize = 0          # 每次交易的数量

# 策略变量
signalPos = {}         # 信号仓位

# 参数列表, 保存了参数的名称
paramList = ['name',
             'className',
             'author',
             'vtSymbol',
             'rsiWindow',
             'rsiLevel',
             'cciWindow',
             'cciLevel',
             'fastWindow',
             'slowWindow',
             'artWindowUnit']
.....

def onBar(self, bar):
    """收到 Bar 推送 (必须由用户继承实现) """
    self.am.updateBar(bar)
    if not self.am.inited:
        return
    self.atrValueUnit = self.am.atr(self.artWindowUnit)

    super(MultiSignalStrategy, self).onBar(bar)
    self.rsiSignal.onBar(bar, self.rsiWindow, self.rsiLevel)
    self.cciSignal.onBar(bar, self.cciWindow, self.cciLevel)
    self.maSignal.onBar(bar)
    self.maSignal.onFiveBar(bar, self.fastWindow, self.slowWindow)
    self.calculateTargetPos()

```



# 第 6 章

## 海龟策略本地化实证

### 6.1 海龟策略速览

#### 6.1.1 海龟策略的故事

海龟策略源于 1983 年两位交易大师理查德·丹尼斯（Richard J. Dennis）与威廉·埃克哈特（William Eckhardt）之间的一次赌约，赌约的内容是伟大的期货交易者能否通过后天的培养产生。当时报名的有 1000 多人，最终有 13 个人参加了为期两周的培训，然后在实战中不断强化其策略执行力。该计划称为“海龟计划”，参与此计划的人被称为“海龟”。“海龟计划”终结于 1987 年，之后“海龟”们都走上了各自的交易或者非交易之路。那些仍然做交易的“海龟”们的交易方式几乎全部采用系统化的、长线的趋势追随策略。《海龟交易法则》作者柯蒂斯·费思（Curtis Faith）是当时最成功的“海龟”，在四年内实现了其他“海龟”三倍的赢利，由此可见，海龟策略必有一些出彩的东西值得研究。

海龟策略严格来说是一个进阶版的适用于投资组合的 CTA 策略，包含入场信号、逐步加仓、单位头寸管理、上一笔赢利过滤、止盈止损等方面，都挺有参考



学习的价值。不过，海龟策略与传统 CTA 策略还是有一些区别的：

- 传统 CTA 策略基于单标的合约（虽然会有相同策略应用于多个品种的情况，但每个策略实例的参数会有些不同），并且其买卖手数都是固定的，每个策略都有其对应的起始资金，投资组合的盈亏曲线也仅仅是每个品种盈亏线的简单累加。故从投资组合的角度看，传统 CTA 策略是一个从局部到整体的过程。
- 海龟策略基于多标的合约，根据总体资金规模及品种的绝对波动幅度，分配到每个品种的手数是不同的，其独特的逐步加仓和单位头寸限制也是传统 CTA 策略所没有的。从投资组合的角度看，海龟策略是一个从整体到局部的过程。

要挖掘出海龟策略出彩的地方，应先复制好完整的海龟策略，验证其在中国市场的有效性，然后逐步解剖海龟策略的关键要素，单独对关键要素进行删除、替换、优化参数等操作，最后提取其精华部分放在传统 CTA 策略上验证。

## 6.1.2 海龟策略的局限性

海龟策略有许多值得学习的地方，但也有其时代的局限性，需要理性看待。因此不要过分神化策略创始人理查德·丹尼斯。尽管丹尼斯曾经风光一时，身家从 400 美元飙升到 2 亿美元，但是经历 1987 年惨败后，旗下基金亏损 50%，他个人资产亏损超过 80%。不久，他的基金在 1988 年 4 月被迫解散，他自己的资产也从全盛时期的 2 亿美元大幅缩水到 1000 多万美元，最后毅然宣布脱离投资市场，从此“金盆洗手”，专心从政（但是他的徒弟们却通过同样的海龟策略大赚特赚）。

究其原因，与人工下单的执行力有关。“海龟”们有丹尼斯作为精神导师，只要市场出现震荡行情，海龟们的账户上都发生了严重的亏损，丹尼斯就出场了，请他们到拉斯维加斯赌场狂欢，并给他们增加资金，告诉他们“亏钱没关系，亏钱是赢钱的成本”“往往一段时间拉锯式的整理行情之后会有疯狂的上涨大行情”。事实正如他所预料的，撑过了困难时期之后，“海龟”们的收益直线上升，平均

收益率达 100% 以上。但是丹尼斯却没有自己的“精神导师”，因此可能溃败于对自己所定规则的反叛。这大概也是为何柯蒂斯·费思（Curtis Faith）在《海龟交易法则》中一而再、再而三地强调交易者要贯彻交易系统的执行能力的原因。

但是随着时代的发展，当初困扰“海龟”们最大的难题现在可以轻松地解决，答案是：只要搭建好程序化交易系统，并且连接好交易用的 API 接口后，一旦接收新行情，模型就会马上进行计算，产生交易信号后也能通过算法下单。

另一个表明“海龟”们“过时”的例子：他们并不用停止单，而是用限价单（不管是入场还是止损离场），害怕让经纪人设置止损指令而泄露策略的头寸，故打电话对不同经纪人报不同的价格。这一缺陷也被程序化交易系统克服了。

纵观历史，丹尼斯及他的“海龟”们并不拔尖。因为在同时期，即 20 世纪 80 年代到 90 年代，爱德华·索普（Edward Thorp）博士的普林斯顿/新港合伙（Princeton/Newport Partner）公司通过可转债套利已经赚得盆满钵满；盈透证券创始人托马斯·彼得菲（Thomas Peterffy）已经可以通过程序化交易（期权做市商）来“收割”其他庄家；詹姆斯·西蒙斯（James Harris Simons）成立的文艺复兴科技公司，在偏高频领域开始了不败传说。对于海龟策略，我们要做的就是取其精华，去其糟粕。

### 6.1.3 原版海龟策略

在《海龟交易法则》最后的附录中概括了原版海龟策略的 7 大要素，如图 6-1 所示。

#### 1. 品种选择

《海龟交易法则》中明确表明海龟策略选择的是流动性高的期货品种，且是具有历史大波动并且无人干预的品种，在中国国内对应的就是四大期货交易所成交量巨大的品种，并且不包括股指期货。



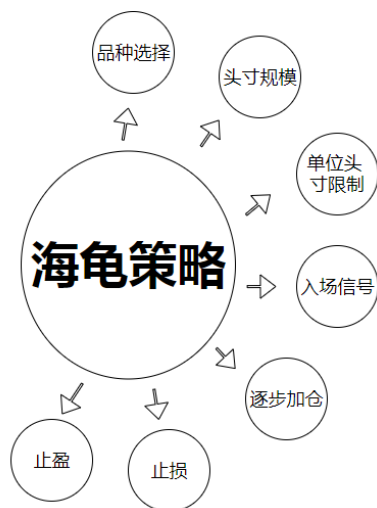


图 6-1 海龟策略的 7 大要素

## 2. 头寸规模

头寸规模是海龟交易系统最重要的部分之一，其优势是根据一个市场的绝对波动幅度来调整头寸规模，等于将头寸的绝对波动幅度标准化。这意味着，一个特定头寸在某一天的向上或向下变动幅度与其他市场的头寸基本相同，无论这个特定市场的波动性是大还是小。

举个例子，若一个市场的波动性较强，则头寸就小一些，反之，若一个市场的波动性较弱，则头寸就可以大一些。总之，市场的波动性与头寸规模是相互抵消的。波动性标准化处理意味着不同交易在盈亏概率上是相同的：它们都有同样的机会赚或赔 1 美元，这便提升了多重市场的分散性。

头寸规模单位公式：

$$\text{头寸规模单位} = \frac{\text{账户的 1\%}}{\text{市场的绝对波动幅度}}$$

或者

$$\text{头寸规模单位} = \frac{\text{账户的 1\%}}{N \times \text{每一点数所代表的美元}}$$

公式中有 2 点需要特别说明一下：“N”和“每一点数所代表的美元”。

- N 表示真实波动幅度 (TR) 的 20 日指数移动平均值, 即 20 日的 ATR。另外需要注意的是, ATR 是基于期货指数得到的, 这意味着在策略实现中, 不管是历史回测还是实盘测试, 都要用到期货指数。解决方法有两个: 要么自己写脚本合成实时的指数行情数据, 要么使用第三方收费的指数行情数据 (考虑开发脚本的难度和运维数据的成本, 显然购买第三方数据性价比高)。
- 每一点数所代表的美元: 每一手合约规模数量, 以美元为计价单位。

原版头寸规模单位计算例子是纽约商品期货交易所 (NYMEX) 民用燃料油 (HO03H) 在 2002 年 12 月 4 日, 日 K 线的最高价、最低价、收盘价分别是 0.7420、0.7140、0.7162, ATR 值是 0.0141, 民用燃料油的合约规模是 42000 加仑 (1000 桶), 若账号资金是 100 万美元, 则得出的头寸规模是 16.88, 四舍五入得头寸规模为 17 手合约。

根据 2002 年 12 月 4 日的 N 值 0.0141, 12 月 6 日的头寸单位规模计算如下:

$$N=0.0141$$

$$\text{账户规模}=1\,000\,000\text{ 美元}$$

每一点数所代表的美元=42 000 (每份合约代表 42 000 加仑民用燃料油, 以美元为计价单位)。

$$\text{头寸单位规模} = \frac{0.01 \times 1\,000\,000}{0.0141 \times 42\,000} = 16.88$$

这里可以看出美国期货合约的规模非常大, 但是价格低 (即 ATR 值小), 这与国内期货合约是相反的。因为国内期货的特点是合约规模小但是价格比较高。下面以螺纹钢合约为例: 2018 年 12 月 12 日, 螺纹钢指数日 K 线的最高价、最低价、收盘价分别是 3416、3356、3356, 20 日的 ATR 值是 103.55, 合约规模是 10



吨, 故其头寸规模  $= (1\% \times 1,000,000) / (103.55 \times 10) = 9.66$ , 四舍五入得 10 手合约。

从上面两个例子可以推断出, 若账号资金不足, 头寸规模调整失去精确性, 则会大大降低风险分散化的效果。

### 3. 单位头寸限制

原版海龟策略通过 4 个层面来限制其成交量, 从而控制交易者的总体风险水平。无论是在没完没了的亏损时期, 还是在翻天覆地的价格动荡中, 这些限制都能将损失最小化。

这四个层面的限制分别如下。

- 单个市场: 头寸上限是 4 个。
- 高度关联的多个市场: 单个方向头寸单位不超过 6 个。
- 松散关联的多个市场: 某一个方向上的头寸单位不超过 10 个。
- 单个方向: 最多 12 个。

这些实际实现还是比较困难的, 因为对于高度关联市场和松散关联市场的判断都非常主观, 并无统一标准, 所以只能简单地实现单个市场和单个方向的头寸限制。

### 4. 入场信号

原版海龟策略提供两个版本的入场信号, 用的都是唐奇安通道<sup>1</sup>突破规则, 其区别在于时间周期不同, 下面分别介绍这两个版本。

---

1 唐奇安通道是由理查德·唐奇安 (Richard Davoud Donchian) 发明的, 唐奇安通道突破规则非常简单: 当价格突破前  $X$  天的最高价时, 做多; 价格突破前  $X$  天的最低价时, 做空。  $X$  值默认为 20, 市场上流行对 20 这个值解释的轶闻是: 唐奇安在开发唐奇安通道期间, 碰巧读到整形外科医生麦克斯威尔·马尔茨 (Maxwell Maltz) 博士在 1960 年所著的《心理控制论》(这本书在 1989 年被重新发现)。马尔茨博士称在整形外科手术过程中, 患者最少需要 21 日来看到自己新的容颜。而很多观察到的现象都显示了最起码需要 21 日来使得新事物代替旧事物。这一事实震惊了唐奇安, 21 个自然日就等于 15 个交易日! 当绝大多数交易者都在认为趋势可能已经变化时 (他们认为看到了市场的新颜), 主要趋势却已做好了继续运行的准备。

- **短周期版本**：若期货指数价格突破 20 日最高价/最低价，则买入/卖出 1 个头寸单位。过滤条件是：上一次突破是赢利性突破，则当前入市信号无效（其保障性突破点为在 55 日通道入市）。若突破日后价格在反方向移动幅度达  $2 \times \text{ATR}$ ，则止损离场。若上一次突破点赢利，那么新突破点可能离当前价远，因为有可能是 55 日突破点；若上一次突破点亏损，那么新突破点将更加接近当前价格。
- **长周期版本**：若价格突破 55 日最高价/最低价，则买入/卖出 1 个头寸单位。无论上一次突破是亏损还是赢利，长周期版本将所有突破都视为有效信号。

“海龟”们可以自由决定在这两个版本之间资金的分配比例，有的“海龟”只用短周期版本，有的则只用长周期版本，也有的各投 50%。

### 5. 逐步加仓

在突破点建立 1 个单位的头寸，然后按  $0.5 \times \text{ATR}$  的价格间隔一步步扩大头寸（ $1/2$  的间隔是以上一份订单的实际成交价格为基础的）。这个过程将继续下去，直到头寸规模达到上限，即单个合约最大 4 个单位头寸。《海龟交易法则》中是以纽约商品交易所（COMEX）黄金期货指数作为例子进行说明的。

黄金：

$$N=2.50$$

$$55 \text{ 日突破价}=310.00$$

$$\text{第一个单位: } 310.00$$

$$\text{第二个单位: } 310.00 + 1/2 \times 2.5 = 311.25$$

$$\text{第三个单位: } 311.25 + 1/2 \times 2.5 = 312.50$$

$$\text{第四个单位: } 312.50 + 1/2 \times 2.5 = 313.75$$



## 6. 止损

止损即价格反方向偏离  $2 \times \text{ATR}$  幅度进行移动止损。但是结合逐步加仓，若在后续中按照  $0.5N$  的价格间隔补充头寸单位，则之前头寸单位的止损点将相应调整  $0.5 \times \text{ATR}$ ，具体例子如下所示。

原油：

$N=1.20$

55 日突破价=28.30

	入市价	止损价
第一个单位	28.30	25.90
	入市价	止损价
第一个单位	28.30	26.50
第二个单位	28.90	26.50
	入市价	止损价
第一个单位	28.30	27.10
第二个单位	28.90	27.10
第三个单位	29.50	27.10

## 7. 止盈

对应入场信号中的短周期版本和长周期版本，止盈策略也分为两个版本。

- **短周期版本**：采用 10 日唐奇安通道突破退出法则，即多头价格跌破 10 日最低点离场，空头价格超过 10 日最高点离场。
- **长周期版本**：采用 20 日唐奇安通道突破退出法则，即多头价格跌破 20 日最低点离场，空头价格超过 10 日最高点离场。

### 6.1.4 策略回测效果

《海龟交易法则》并没有展示完整版海龟策略的历史回测效果图，仅仅提供了简化版（作者称之为“唐奇安趋势系统”）。费思通过交易模拟软件 TradingBlox



Builder 对其进行回测，回测区间设置为 1996 年 1 月到 2006 年 6 月，回测相关指标如表 6-1 所示，资金曲线如图 6-2 所示。

从简化版可以推测海龟策略整体上是赢利的，尽管中途有比较大的资金回撤。其夏普比率接近 1，百分比最大回撤和年化收益都比较高，故属于高风险、高收益类型；胜率低，靠少数大行情的赢利可以弥补大部分假突破所造成的亏损，表现出趋势跟踪策略的典型特点。

表 6-1 简化版海龟策略回测相关指标

系统	CAGR（平均复合增长率）	MAR 比率	夏普比率	交易次数	交易成功率	最大衰落	衰落持续期
ATR 通道突破	49.5%	1.24	1.34	206	42.2%	39.9%	8.3
布林格通道突破	51.8%	1.52	1.52	130	54.6%	34.1%	7.8
唐奇安趋势	29.4%	0.80	0.99	1832	39.7%	36.7%	27.6
唐奇安定时	57.2%	1.31	1.35	746	58.3%	43.6%	12.1
双重均线	57.8%	1.82	1.55	210	39.5%	31.8%	8.3
三重均线	48.1%	1.53	1.37	181	42.5%	31.3%	8.5

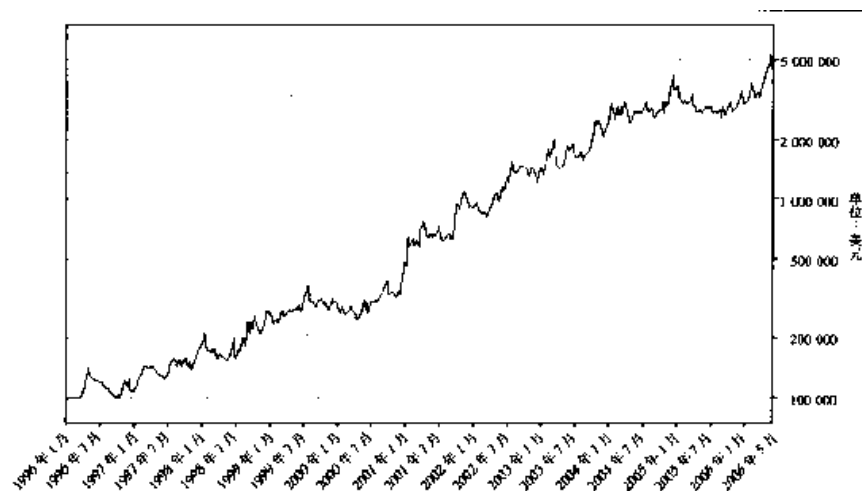


图 6-2 简化版海龟策略资金曲线



## 6.2 本地化实现困境与解决方案

### 6.2.1 本地化实现困境

海龟策略如此出名，但当前国内还没有发现能够完全复制海龟策略的，普遍存在的问题如下：

- 原版海龟策略用的是日线级别数据，但还是要求 K 线内成交，即前一日确定第二天的入场、出场、止损位置，当价格到达指定位置时，立即发出交易委托，而非基于股票多因子框架以收盘价成交。
- 原版海龟策略用的是期货指数的投资组合数据，但是国内回测要么基于股票，要么基于基金，并且都只是针对单标的的回测。
- 在期货上运行原版海龟策略意味着既能做多，又能做空；但是对于股票品种，做空的难度很大，即融券难。其难度表现在资金门槛高，能融的券商少，融券需要付出高昂的利息等。
- 原版海龟策略的入场信号和止盈是短周期版本和长周期版本结合的；由于短周期版本在国内实现困难（难点在于实现过滤条件：若上一次突破是赢利性突破，则当前入场信号无效），所以中国全部采用的是长周期版本。

原版海龟策略尽管有比较大的回撤，但是整体上资金曲线是向上的。与之相反，国内复制的海龟策略回测结果显得有点不尽如人意，要么回撤大，夏普比率低；要么曲线显得非常奇怪。回测效果如图 6-3 所示。

从本质上看，原版海龟策略是一个高风险、高收益、基于投资组合的中低频趋势跟踪策略，并不复杂，但是国内本地化复制完成度低，究其原因可能是：散户资金量不足，编程水平不高，没有能力实现；私募机构则不愿意承担这么大的回撤风险，怕吓跑客户，故不愿意研究；自营机构或许已经通过海龟策略赚大钱，不愿意公布。

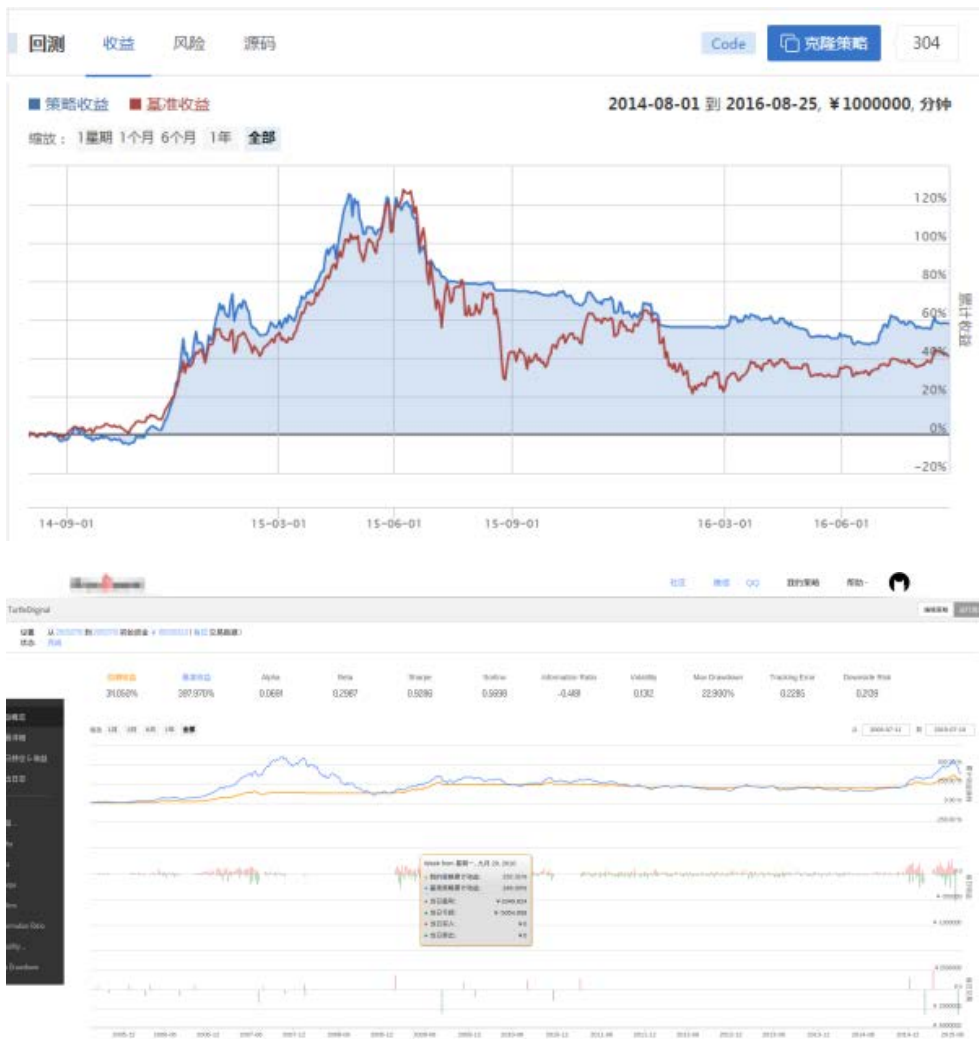


图 6-3 原版海龟策略回测效果



图 6-3 原版海龟策略回测效果 (续)

### 6.2.2 理想解决方案

vn.py 1.9.1 新增完整的投资组合级别的海龟策略实现，经过多次测试发现，这一次海龟策略本地化实现的完成度很高。其投资组合回测资金曲线如图 6-4 所示。投资品种分别是铝、铜、螺纹钢、锌、普麦、一号棉花、玉米、铁矿石、焦煤、焦炭、豆粕、聚氯乙烯。回测时间是 2014—2018 年，百分比最大回撤是 -29.46%，年化收益为 45.11%，夏普比率达 1.5。从资金曲线的形态上看，虽然时不时发生回撤，但是整体趋势是向上的。故从回测上看，已经非常符合原版海龟策略的特点了（因为考虑到海龟策略是日 K 线中低频策略，故手续费和滑点在回测中都设置成 0，以方便操作。但就算加上手续费和滑点，资金曲线的形态也不会发生变化，仅仅是夏普比率稍微减少）。

首个交易日: 2014-01-02 00:00:00  
最后交易日: 2018-12-12 00:00:00  
总交易日: 1208  
盈利交易日: 591  
亏损交易日: 557  
起始资金: 10000000  
结束资金: 77,530,432.94  
总收益率: 675.3%  
年化收益: 45.11%  
总盈亏: 67,530,432.93  
最大回撤: -9,105,393.35  
百分比最大回撤: -29.46%  
总手续费: 0.0  
总滑点: 0.0  
总成交笔数: 1,143.0  
日均盈亏: 55,902.68  
日均手续费: 0.0  
日均滑点: 0.0  
日均成交笔数: 0.0  
日均收益率: 0.19%  
收益标准差: 1.94%  
Sharpe Ratio: 1.5



图 6-4 投资组合回测资金曲线

## 6.3 vn.py 实现的海龟策略

### 6.3.1 工具准备

主要需要准备以下两个工具。

- **RQData**: RQData 是 RiceQuant (米筐科技) 提供的商用版金融数据工具包, 支持 Python、MATLAB、Excel 插件等多种访问方式。它集成了简单高效的 API 接口, 用户可快速调用丰富整齐的量化金融数据, 最大限度地免除数据搜索、清洗的烦恼, 加速研发及投资的决策周期。RQData 期货数据终端提供 7 天免费试用。
- **vn.py**: vn.py 是基于 Python 语言的量化交易系统, 其独特的事件驱动引擎和逐条数据回放的回测设计, 杜绝未来函数的可能性。同时, 回测引擎和实盘引擎设计采用了完全兼容的 API 函数, 用户可以使用同一套策略代码实现回测研究和执行实盘交易 (1.9.1 版本以后提供海龟策略模块, 老版本建议升级, 升级方法: 先用命令 `pip uninstall vnp` 卸载 vn.py, 再安装新版本)。

## 6.3.2 数据准备

### 1. 安装 RQData

输入下面命令可在 Windows 系统上安装 RQData。

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple --extra-index-url
https://rquser:ricequant99@py.r
icequant.com/simple/ rqdatac==1.0.0a29
```

### 2. 下载数据

在“examples\DataService\RqdataDataService”文件夹内的 config.json 上填写 RQData 的账号和密码，保存退出后启动 downloadData.py 下载期货全品种的日线级别数据。启动操作是用“Shift 键+鼠标右键”选择“在此处打开命令窗口”，然后输入下面命令即可。

```
python downloadData.py
```

本次投资组合回测时间区间是 2014 年 1 月 1 日以后的数据，故在该回测上会剔除比较新的品种：

```
"""
#立即下载数据到数据库中，用于手动执行更新操作
"""
from dataService import *
if __name__ == '__main__':

    #中金所
    downloadDailyBarBySymbol('IF99')
    downloadDailyBarBySymbol('IC99')    #中证 500 从 2015 年开始
    downloadDailyBarBySymbol('IH99')    #上证 50 从 2015 年开始

    #上期所
    downloadDailyBarBySymbol('CU99')
    downloadDailyBarBySymbol('AL99')
    downloadDailyBarBySymbol('ZN99')
    downloadDailyBarBySymbol('PB99')
    downloadDailyBarBySymbol('NI99')    #镍从 2015 年开始
    downloadDailyBarBySymbol('AU99')
```

```
downloadDailyBarBySymbol('SN99') #锡从 2015 年开始
downloadDailyBarBySymbol('AG99')
downloadDailyBarBySymbol('RB99')
downloadDailyBarBySymbol('WR99')
downloadDailyBarBySymbol('HC99') #热轧卷板从 2014 年 3 月开始
downloadDailyBarBySymbol('SC99') #燃油从 2018 年开始
downloadDailyBarBySymbol('BU99')
downloadDailyBarBySymbol('RU99')
```

#大商所

```
downloadDailyBarBySymbol('C99')
downloadDailyBarBySymbol('CS99') #玉米淀粉从 2014 年 12 月开始
downloadDailyBarBySymbol('A99')
downloadDailyBarBySymbol('B99')
downloadDailyBarBySymbol('M99')
downloadDailyBarBySymbol('Y99')
downloadDailyBarBySymbol('P99')
downloadDailyBarBySymbol('FB99')
downloadDailyBarBySymbol('BB99')
downloadDailyBarBySymbol('JD99')
downloadDailyBarBySymbol('L99')
downloadDailyBarBySymbol('V99')
downloadDailyBarBySymbol('PP99') #聚丙烯从 2014 年 2 月开始
downloadDailyBarBySymbol('J99')
downloadDailyBarBySymbol('JM99')
downloadDailyBarBySymbol('I99')
```

#郑商所

```
downloadDailyBarBySymbol('TA99')
downloadDailyBarBySymbol('MA99') #甲醇从 2014 年 6 月开始
downloadDailyBarBySymbol('FG99')
downloadDailyBarBySymbol('SF99')
downloadDailyBarBySymbol('SM99')
downloadDailyBarBySymbol('ZC99') #动力煤从 2015 年开始
downloadDailyBarBySymbol('WH99')
downloadDailyBarBySymbol('PM99')
downloadDailyBarBySymbol('CF99')
downloadDailyBarBySymbol('SR99')
downloadDailyBarBySymbol('OI99')
downloadDailyBarBySymbol('RI99')
downloadDailyBarBySymbol('RS99')
```



```
downloadDailyBarBySymbol('RM99')
downloadDailyBarBySymbol('JR99')
downloadDailyBarBySymbol('LR99')    #晚籼稻从 2014 年 7 月开始
downloadDailyBarBySymbol('CY99')    #棉花从 2017 年开始
downloadDailyBarBySymbol('AP99')    #苹果从 2017 年开始
```

数据成功下载界面如图 6-5 所示。

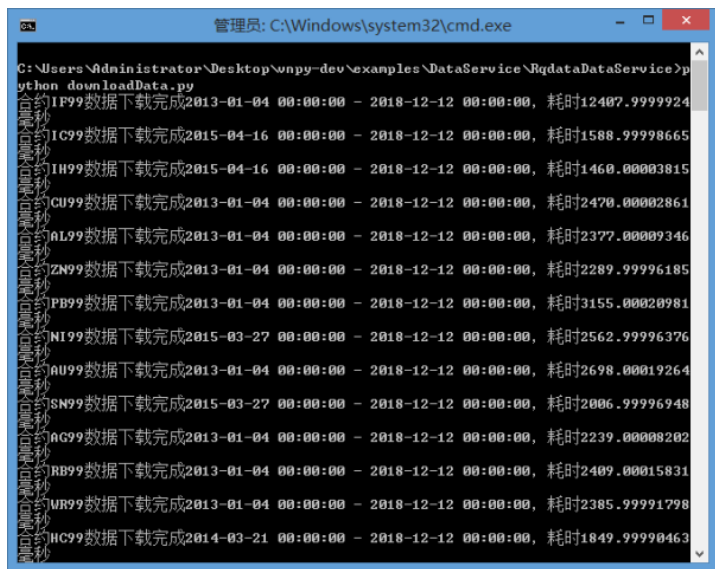


图 6-5 使用 RQData 下载期货全品种日线数据

### 3. 数据说明

RQData数据分别是以 88 或 888 结尾的代表主力连续合约<sup>1</sup>数据和以 99 结尾的指数连续合约数据。

- IF88: 主力连续合约, 由 IF 股指期货不同时期主力合约接续而成, 仅仅是合约量价数据的简单拼接, 未做“平滑”处理。
- IF888: 主力连续合约, 对价格拼接进行了“平滑”处理, 即以主力合约切换

1 主力连续合约是指合约首次上市时, 以当日收盘同品种持仓量最大者作为从第二个交易日开始的主力合约。当同品种其他合约持仓量在收盘后超过当前主力合约 1.1 倍时, 从第二个交易日开始进行主力合约的切换。日内不会进行主力合约的切换。



前一天（T-1 日）新、旧两个主力合约收盘价做差，之后将 T-1 日及以前的主力连续合约的所有价格水平整体加上或减去该价差，以“整体抬升”或“整体下跌”主力合约的价格水平，成交量、持仓量均不调整，成交额统一设置为 0。

- IF99：指数连续合约，由 IF 股指期货全部可交易合约以累计持仓量为权重加权平均得到的。

### 6.3.3 海龟策略代码结构

在运行策略回测前，首先要对策略代码有一个整体的概念。vn.py 1.9.1 提供了两个版本的海龟策略，分别是基于 ctaTemplate 开发的、针对单标的的简化版，以及针对多标的的完整版。

这里只介绍完整版的海龟策略，该模块在“examples\TurtleStrategy”文件夹下，如图 6-6 所示，这里只需关注 4 个文件。

- setting.csv：用于设置策略回测所用到的数据，包括期货品种、合约规模、最小价格变动、手续费、滑点。
- turtleStrategy.py：具体的海龟策略。
- turtleEngine.py：海龟策略回测引擎。
- run.ipynb：在 Jupyter Notebook 上进行回测。

 .ipynb_checkpoints	2018/12/2 星期...	文件夹	
 000300.csv	2018/12/2 星期...	Microsoft Excel ...	83 KB
 000905.csv	2018/12/2 星期...	Microsoft Excel ...	83 KB
 loadCsv.py	2018/12/2 星期...	PY 文件	1 KB
 README.MD	2018/12/2 星期...	Markdown File	1 KB
 run.ipynb	2018/12/2 星期...	IPYNB 文件	3 KB
 setting.csv	2018/12/2 星期...	Microsoft Excel ...	1 KB
 test.csv	2018/12/2 星期...	Microsoft Excel ...	1 KB
 turtleEngine.py	2018/12/2 星期...	PY 文件	16 KB
 turtleStrategy.py	2018/12/2 星期...	PY 文件	15 KB

图 6-6 TurtleStrategy 内文件

现在只关注 `turtleStrategy.py` 这个文件。海龟策略由 3 个类构成，分别是 `TurtleResult`、`TurtleSignal`、`TurtlePortfolio`。阅读顺序并不是简单的从上到下，而是镶嵌结构，故一开始并不容易读懂，海龟策略的代码结构如图 6-7 所示，箭头方向代表先定义，后调用。

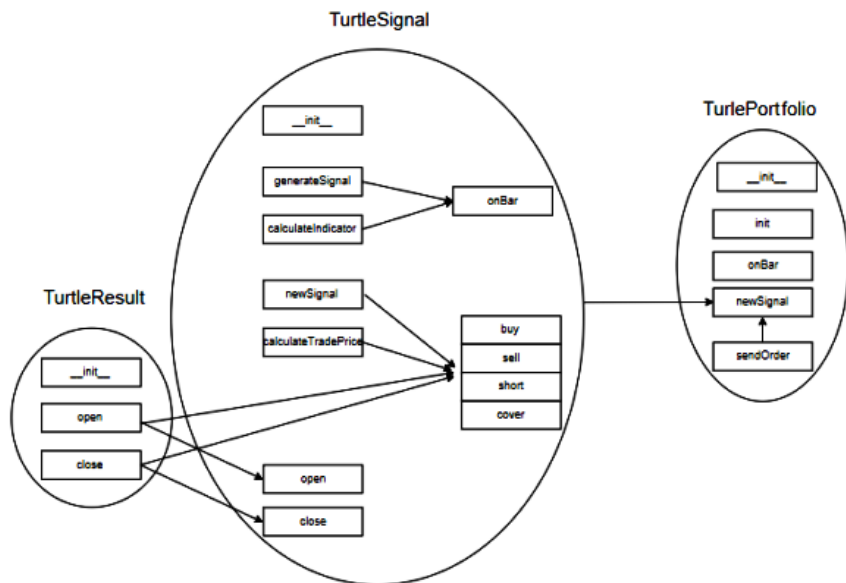


图 6-7 海龟策略的代码结构

## 1. TurtleResult 类

用于计算单笔开平仓交易盈亏，是海龟策略中判断“上一笔赢利当前信号是否无效”的基础，其函数如下。

- `__init__()`：初始化单位头寸、开仓均价、平仓均价和单笔开平仓交易盈亏数。
- `open()`：先计算开仓累计成本，然后统计开仓平均成本。
- `close()`：缓存平仓均价，统计单笔开平仓交易盈亏。

## 2. TurtleSignal 类

用于产生海龟策略交易信号，包括入场、止损、止盈委托价格与目标仓位，

其函数如下所示。

- `__init__()`: 初始化海龟信号的策略参数（默认不检查上一笔盈亏，以及缓存 60 根 K 线）。
- `onBar()`: 缓存足够 K 线后，开始计算相关技术指标，判断交易信号。
- `generateSignal()`: 负责交易信号的判断，平仓信号与开仓信号是分开的，优先检查平仓，没有仓位或者持有多头仓位的时候，设置好入场位做多或加仓；没有仓位或者持有空头仓位的时候，设置好入场位做空或者加仓。
- `calculateIndicator`: 负责计算指标的产生，包括计算入场和止盈离场的唐奇安通道上下轨，判断到有单位持仓后，计算 ATR 指标并且设定随后 8 个入场位置（做多 4 个和做空 4 个），同时初始化离场价格。
- `newSignal()`: 定义海龟投资组合的发单委托，分别是多空方向、开仓平仓、停止单价格、合约手数。
- `buy()`: 传入计算好的停止单价格，缓存开仓委托的价格和手数，发出投资组合的多开委托，基于最后一次加仓价格计算止损离场位置。
- `sell()`: 传入计算好的停止单价格，缓存平仓委托的价格，发出投资组合的空平委托。
- `short()`: 传入计算好的停止单价格，缓存开仓委托的价格和手数，发出投资组合的空开委托，基于最后一次加仓价格计算止损离场位置。
- `cover()`: 传入计算好的停止单价格，缓存平仓委托的价格，发出投资组合的多平委托。
- `open()`: 计算累计开仓手数/单位头寸，调用 `TurtleResult` 类定义的 `open` 函数计算开仓平均成本。
- `close()`: 调用 `TurtleResult` 类定义的 `close` 函数计算单笔开平仓交易盈亏。创建列表，专门缓存开平仓交易盈亏。
- `getLastPnl()`: 在开平仓交易盈亏列表中获取上一笔交易的盈亏。
- `calculateTradePrice()`: 设置停止单价格，要求买入时，停止单成交的最优价格不能低于当前 K 线开盘价；卖出时，停止单成交的最优价格不能高于当前 K 线开盘价。



### 3. TurtlePortfolio 类

根据账户资金和品种合约规模，产生具体交易委托，其函数如下所示。

- `__init__()`: 初始化海龟投资组合的组合市值（即账户资金）和多空头持仓，创建多个字典分别缓存海龟信号、每个品种持仓情况、交易中的信号、合约大小、单位头寸规模、真实持仓量。
- `init()`: 传入组合市值和合约大小字典，调用 `TurtleSignal` 类产生短周期版本和长周期版的交易信号（包括入场、止盈、止损），同时缓存到信号字典中。
- `onBar()`: 根据信号字典产生具体交易委托。
- `newSignal()`: 先计算单位头寸规模，若委托指令是开仓，则需要检查上一次是否赢利，无赢利就发出买入/卖空委托；若委托指令是平仓，则需要注意平仓量不能超过空头持仓，同时注意单品种和组合持仓都不能超过上限。
- `sendOrder()`: 计算单品种持仓和整体持仓，向回测引擎中发单记录。

## 6.3.4 海龟策略 6 大要素代码解析

海龟策略 7 大要素分别是品种选择、头寸规模、单位头寸限制、入场信号、逐步加仓、止损、止盈。由于品种选择不属于交易策略内容，故只对后面 6 大要素进行代码解析。

### 1. 头寸规模

头寸规模 =  $(1\% \times \text{账户资金}) / (\text{ATR} \times \text{合约规模})$ ，其代码如下：

```
class TurtlePortfolio(object):
    .....
    #-----
    def newSignal(self, signal, direction, offset, price, volume):
    .....
        riskValue = self.portfolioValue * 0.01
        multiplier = riskValue / (signal.atrVolatility * size)
        multiplier = int(round(multiplier, 0)) # multiplier 即为头寸规模
        self.multiplierDict[signal.vtSymbol] = multiplier
    .....
```



```

#-----
def sendOrder(self, vtSymbol, direction, offset, price, volume, multiplier):
    """
    # 计算合约持仓
    if direction == DIRECTION_LONG:
        self.unitDict[vtSymbol] += volume      # volume 即为单位头寸
        self.posDict[vtSymbol] += volume * multiplier
    else:
        self.unitDict[vtSymbol] -= volume
        self.posDict[vtSymbol] -= volume * multiplier # 实际持仓=单位头寸 × 头寸
#规模

```

## 2. 单位头寸限制

基于高度关联市场和松散关联市场判断起来都非常主观，并无统一标准。客观上只能实现单个市场和单个方向的头寸限制。

在开仓交易前，需要检查上一笔交易是否赢利，若赢利则直接返回，不进行买卖操作（仅仅适用于短周期版本的入场策略，即 profitCheck=True；长周期版本对应的是 profitCheck=False）。

```

MAX_PRODUCT_POS = 4      # 单品种最大持仓
MAX_DIRECTION_POS = 10   # 单方向最大持仓
.....
class TurtlePortfolio(object):
    .....
    #-----
    def newSignal(self, signal, direction, offset, price, volume):
    .....
        # 开仓
        if offset == OFFSET_OPEN:
            # 检查上一次是否为赢利
            if signal.profitCheck:
                pnl = signal.getLastPnl()
                if pnl > 0:
                    return

        # 买入
        if direction == DIRECTION_LONG:
            # 组合持仓不能超过上限
            if self.totalLong >= MAX_DIRECTION_POS:

```



```

        return

    # 单品种持仓不能超过上限
    if self.unitDict[signal.vtSymbol] >= MAX_PRODUCT_POS:
        return

    # 卖出
    else:
        if self.totalShort <= -MAX_DIRECTION_POS:
            return

        if self.unitDict[signal.vtSymbol] <= -MAX_PRODUCT_POS:
            return

```

### 3. 入场信号、逐步加仓、止损、止盈

原版海龟策略提供两个版本的入场和止盈信号，分别是长周期版本和短周期版本的唐奇安通道突破规则。

- 短周期版本：入场用的是 20 日唐奇安通道，止盈用的是 10 日唐奇安通道，用 20 日周期计算 ATR 值，有上一笔赢利当前信号无效的过滤条件。
- 长周期版本：入场用的是 55 日唐奇安通道，止盈用的是 20 日唐奇安通道，用 20 日周期计算 ATR 值，无上一笔赢利当前信号无效的过滤条件。

逐步加仓的规则是每隔  $0.5 \times \text{ATR}$  幅度慢慢加满至 4 个单位头寸，止损也相应根据  $0.5 \times \text{ATR}$  的步进移动。

```

class TurtleSignal(object):
    #-----
    def __init__(self, portfolio, vtSymbol,
                 entryWindow, exitWindow, atrWindow,
                 profitCheck=False):
    .....
    #-----
    def onBar(self, bar):
        self.bar = bar
        self.am.bar
        if not self.am.inited:
            return

        self.generateSignal(bar)

```

```
self.calculateIndicator()
#-----
def generateSignal(self, bar):
    """
    判断交易信号
    """
    # 如果指标尚未初始化, 则忽略
    if not self.longEntry1:
        return

    # 优先检查平仓
    if self.unit > 0:
        longExit = max(self.longStop, self.exitDown)

        if bar.low <= longExit:
            self.sell(longExit)
            return
    elif self.unit < 0:
        shortExit = min(self.shortStop, self.exitUp)
        if bar.high >= shortExit:
            self.cover(shortExit)
            return

    # 没有仓位或者持有多头仓位时, 可以做多(加仓)
    if self.unit >= 0:
        trade = False

        if bar.high >= self.longEntry1 and self.unit < 1:
            self.buy(self.longEntry1, 1)
            trade = True

        if bar.high >= self.longEntry2 and self.unit < 2:
            self.buy(self.longEntry2, 1)
            trade = True

        if bar.high >= self.longEntry3 and self.unit < 3:
            self.buy(self.longEntry3, 1)
            trade = True

        if bar.high >= self.longEntry4 and self.unit < 4:
            self.buy(self.longEntry4, 1)
            trade = True
```



```

        if trade:
            return

# 没有仓位或者持有空头仓位时, 可以做空 (加仓)
if self.unit <= 0:
    if bar.low <= self.shortEntry1 and self.unit > -1:
        self.short(self.shortEntry1, 1)

    if bar.low <= self.shortEntry2 and self.unit > -2:
        self.short(self.shortEntry2, 1)

    if bar.low <= self.shortEntry3 and self.unit > -3:
        self.short(self.shortEntry3, 1)

    if bar.low <= self.shortEntry4 and self.unit > -4:
        self.short(self.shortEntry4, 1)

#-----
def calculateIndicator(self):
    """计算技术指标"""
    self.entryUp, self.entryDown = self.am.donchian(self.entryWindow)
    self.exitUp, self.exitDown = self.am.donchian(self.exitWindow)

# 有持仓后, ATR 波动率和入场位等都不再变化
if not self.unit:
    self.atrVolatility = self.am.atr(self.atrWindow)

    self.longEntry1 = self.entryUp
    self.longEntry2 = self.entryUp + self.atrVolatility * 0.5
    self.longEntry3 = self.entryUp + self.atrVolatility * 1
    self.longEntry4 = self.entryUp + self.atrVolatility * 1.5
    self.longStop = 0

    self.shortEntry1 = self.entryDown
    self.shortEntry2 = self.entryDown - self.atrVolatility * 0.5
    self.shortEntry3 = self.entryDown - self.atrVolatility * 1
    self.shortEntry4 = self.entryDown - self.atrVolatility * 1.5
    self.shortStop = 0

.....
class TurtlePortfolio(object):

```



```

.....
#-----
def init(self, portfolioValue, vtSymbolList, sizeDict):
    """
    self.portfolioValue = portfolioValue
    self.sizeDict = sizeDict

    for vtSymbol in vtSymbolList:
        signal1 = TurtleSignal(self, vtSymbol, 20, 10, 20, True)
        signal2 = TurtleSignal(self, vtSymbol, 55, 20, 20, False)

        l = self.signalDict[vtSymbol]
        l.append(signal1)
        l.append(signal2)
        self.unitDict[vtSymbol] = 0
        self.posDict[vtSymbol] = 0

#-----
def onBar(self, bar):
    """
    for signal in self.signalDict[bar.vtSymbol]:
        signal.onBar(bar)
.....

```

### 6.3.5 海龟策略的回测

#### 1. 运行 run.ipynb 文件

首先进入“examples\TurtleStrategy”文件夹，在 Jupyter Notebook 中打开 run.ipynb 即可执行策略回测。

##### (1) 调用海龟回测引擎。

```

%matplotlib inline
from datetime import datetime
import Numpy as np
import matplotlib.pyplot as plt
from turtleEngine import BacktestingEngine

```

##### (2) 设置回测时间区间和起始资金，读取 csv 文件的合约信息进行策略回测，



然后显示逐日统计的相关指标和资金图。

```
engine = BacktestingEngine()
engine.setPeriod(datetime(2014, 1, 1), datetime(2018, 12, 30))
engine.initPortfolio('setting.csv', 10000000)

engine.loadData()
engine.runBacktesting()
engine.showResult()
```

(3) 针对投资组合里的单个品种, 查阅其逐步开平仓记录。

```
tradeList = engine.getTradeData('J99')
for trade in tradeList:
    print '%s %s %s %s@%s' %(trade.vtSymbol, trade.direction, trade.offset,
                              trade.volume, trade.price)
```

## 2. 配置 JSON 文件

在运行海龟策略回测时会读取同一文件夹内的 csv 文件, 下面以 setting.csv 为例说明一下, 如图 6-8 所示。

```
1 vtSymbol,size,priceTick,variableCommission,fixedCommission,slippage
2 IF99,300,0.2,0.00003,0,0.2
3 I99,100,0.5,0.00006,0,0.5
4 CU99,5,10,0.00005,0,10
5 TA99,5,2,0,12,2
6
```

图 6-8 JSON 文件内容

需要配置的合约信息包括合约品种、合约规模、最小价格变动、手续费率(如每一手 0.00003)、固定手续费(如每一手 12 元)、滑点。其中手续费率与固定手续费可二选一。以 PTA 合约为例, 其品种信息为 TA99<sup>1</sup>, 合约规模是 5 吨, 最小价格变动是 2 元/吨, 手续费率为 0, 固定手续费为 12 元, 滑点为 2 元。

1 《海龟交易法则》明确表示其交易信号源于期货指数合约, 故用“99”结尾的 RQData 合约进行策略回测。

## 6.4 品种选择验证

### 6.4.1 原版投资组合测试

原版海龟策略选择标准主要是流动性强,若简单地理解为交易所成交量巨大的热门品种,则根据交易所分类所构建的组合历史回测如图 6-9 所示(测试环境是无手续费、无滑点)。图 6-9 中显示,上期所热门品种组合夏普比率达到 1.01,郑商所的达到 0.8,大商所的达到 1.31,而中金所因为只有 IF 股指期货成交量较高,故中金所只测试了一个品种,其夏普比率达到 0.88。总体来看,原版海龟测试夏普比率都不错,有一定的稳健性。

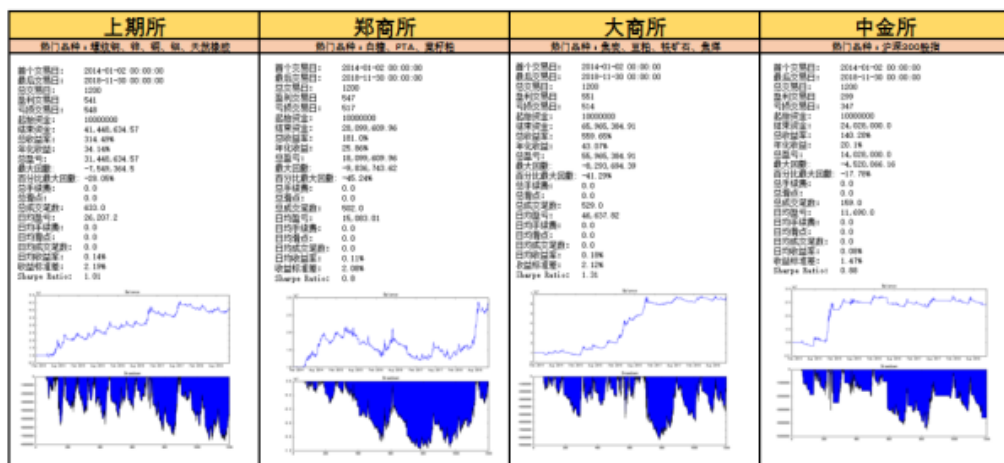


图 6-9 四大交易所热门品种组合历史回测

因为国内四大交易所包含了金融产品、工业品、农业品、金属、化工等不同品种分类,为了分散投资组合各个头寸的风险,提高组合的夏普比率,故海龟策略投资组合品种必须涵盖四大交易所的品种,现在简单地把四大交易所热门品种组合起来进行测试,其效果如图 6-10 所示。

```

首个交易日: 2014-01-02 00:00:00
最后交易日: 2018-12-12 00:00:00
总交易日: 1208
盈利交易日: 581
亏损交易日: 562
起赔资金: 10000000
结束资金: 69,894,435.0
总收益率: 598.94%
年化收益: 43.91%
总盈亏: 59,894,435.0
最大回撤: -10,465,905.94
百分比最大回撤: -29.84%
总手续费: 0.0
总滑点: 0.0
总成交笔数: 1,166.0
日均盈亏: 49,581.49
日均手续费: 0.0
日均滑点: 0.0
日均成交笔数: 0.0
日均收益率: 0.18%
收益标准差: 2.11%
Sharpe Ratio: 1.34

```

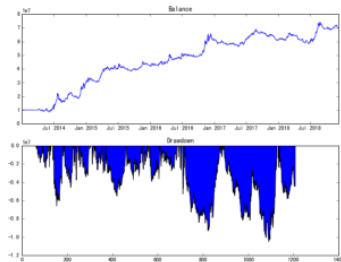


图 6-10 热门品种组合测试结果

新组合的夏普比率达到 1.34，要高于上面四个组合，年化收益 43.91%，百分比最大回撤达到 -29.84%，表现出高风险、高收益的特点，与原版海龟策略基本吻合。

当前品种选择的检验非常顺利，现在只有一个问题：能否在新组合中继续筛选，剔除表现不好的品种，去构建一个具有更高夏普比率的组合呢？答案是否定的。这是一个思维误区，对过去历史表现进行优化，然后筛选出拟合历史行情最优品种，显然没有注意到由未来函数导致的过拟合的问题。

## 6.4.2 筛选品种的传统方法

传统意义上，为保证策略的稳健性，应该进行样本内外检验，其具体步骤如下：

- (1) 设定好历史回测周期，如 2 年的回望周期。

(2) 设置固定的筛选标准, 如夏普比率大于 0.6。

(3) 基于筛选标准, 对每个品种进行策略回测, 然后选择合格的品种组成投资组合。

(4) 通过样本内筛选出来的组合, 去预测其未来表现, 如预测 1 年的策略表现。

(5) 回望周期向后滚动, 基于新组合再次筛选品种, 直到预测最新年份未知。

### 1. 初步筛选

仅仅基于历史行情筛选, 本次测试规定历史回测时间从 2014 年 1 月 1 日开始, 故首先剔除在该时间点后才上市的品种, 进行历史回测的品种从 51 个降低至 35 个。一些如苹果等非常热门的新品种, 不在本次测试范围内。

### 2. 滚动回测

滚动回测标准如下所示。

- 回望周期: 2 年。
- 夏普比率标准:  $>0.6$ 。
- 回望周期步进: 1 年。

#### (1) 2014—2015 年测试

对 35 个品种进行海龟策略的历史回测, 如图 6-11 所示(图 6-11 中的小字“夏普统计出错”, 其意思是总收益为负数, 有时候转换成年化收益变成正数, 导致负的夏普比率变成正数)。



[illegible][illegible][illegible][illegible]

图 6-11 全品种 2014—2015 年历史回测

Q999		Q999		Q999		Q999		Q999		Q999	
2014-2015 历史表统计		2014-2015 历史表统计		2014-2015 历史表统计		2014-2015 历史表统计		2014-2015 历史表统计		2014-2015 历史表统计	
最小交易日期:	2014-01-02 00:00:00	最小交易日期:	2014-01-02 00:00:00	最小交易日期:	2014-01-02 00:00:00	最小交易日期:	2014-01-02 00:00:00	最小交易日期:	2014-01-02 00:00:00	最小交易日期:	2014-01-02 00:00:00
最大交易日期:	2014-12-30 00:00:00	最大交易日期:	2014-12-30 00:00:00	最大交易日期:	2014-12-30 00:00:00	最大交易日期:	2014-12-30 00:00:00	最大交易日期:	2014-12-30 00:00:00	最大交易日期:	2014-12-30 00:00:00
总交易笔数:	480	总交易笔数:	480	总交易笔数:	480	总交易笔数:	480	总交易笔数:	480	总交易笔数:	480
总成交金额:	1100	总成交金额:	105	总成交金额:	100	总成交金额:	100	总成交金额:	100	总成交金额:	100
平均成交金额:	2.2916666666666666	平均成交金额:	0.21875	平均成交金额:	0.20833333333333334	平均成交金额:	0.20833333333333334	平均成交金额:	0.20833333333333334	平均成交金额:	0.20833333333333334
成交笔数分布:	6,939,208.02	成交笔数分布:	7,212,308.0	成交笔数分布:	-13,610,970.0	成交笔数分布:	4,793,570.0	成交笔数分布:	13,961,200.0	成交笔数分布:	13,961,200.0
成交金额分布:	-90.418	成交金额分布:	-57.828	成交金额分布:	39.218	成交金额分布:	39.218	成交金额分布:	3,961,200.0	成交金额分布:	3,961,200.0
最大单笔交易:	-14.078	最大单笔交易:	-13.648	最大单笔交易:	-1.0431308	最大单笔交易:	-0.87	最大单笔交易:	16.184	最大单笔交易:	16.184
最小单笔交易:	-3,346,903.14	最小单笔交易:	-5,302,362.000	最小单笔交易:	-10,610,970.000	最小单笔交易:	3,961,200.000	最小单笔交易:	-3,311,848.87	最小单笔交易:	-3,311,848.87
日均成交金额:	-90.418	日均成交金额:	-49,975.655	日均成交金额:	39.218	日均成交金额:	-10,242.246	日均成交金额:	16.184	日均成交金额:	16.184
日均交易笔数:	0.0	日均交易笔数:	0.0	日均交易笔数:	0.0	日均交易笔数:	0.0	日均交易笔数:	0.0	日均交易笔数:	0.0
日均成交金额:	72.0	日均成交金额:	77.0	日均成交金额:	90.0	日均成交金额:	99.0	日均成交金额:	54.0	日均成交金额:	54.0
日均交易笔数:	-67.129	日均交易笔数:	-66,302.668	日均交易笔数:	66,302.668	日均交易笔数:	66,302.668	日均交易笔数:	66,302.668	日均交易笔数:	66,302.668
日均成交金额:	0.0	日均成交金额:	0.0	日均成交金额:	0.0	日均成交金额:	0.0	日均成交金额:	0.0	日均成交金额:	0.0
日均交易笔数:	0.0	日均交易笔数:	0.0	日均交易笔数:	0.0	日均交易笔数:	0.0	日均交易笔数:	0.0	日均交易笔数:	0.0
日均成交金额:	0.0	日均成交金额:	0.0	日均成交金额:	0.0	日均成交金额:	0.0	日均成交金额:	0.0	日均成交金额:	0.0
日均交易笔数:	-0.008	日均交易笔数:	-0.008	日均交易笔数:	0.008	日均交易笔数:	0.008	日均交易笔数:	0.008	日均交易笔数:	0.008
日均成交金额:	1.414	日均成交金额:	1.414	日均成交金额:	00.099	日均成交金额:	36.394	日均成交金额:	1.078	日均成交金额:	1.078
日均交易笔数:	-0.59	日均交易笔数:	-0.59	日均交易笔数:	-0.59	日均交易笔数:	-0.48	日均交易笔数:	0.96	日均交易笔数:	0.96

2014-2015 年度資料

最新公開日期:	2014-01-02 00:00:00
最新估值日期:	2009-12-30 00:00:00
公司名稱:	800
證券類別:	SI
上市日期:	8/8
起計淨值:	100000000
起計淨值(美元):	-17,793,240.0
起計淨值(港元):	-6877.87%
淨值(美元):	231.65%
淨值(港元):	-8777,240.0
淨值(美元)大數:	-87,987,507.41
淨值(港元)大數:	-693.00%
公司總值:	0
公司總值(美元):	157.0
公司總值(港元):	-179,881.89%
淨資本總額:	0
淨資本總額(美元):	0
淨資本總額(港元):	0.07%
淨資本總額大數:	0
淨資本總額大數(美元):	62.43%
Sharpe Ratio:	0.24

[illegible]

时段	P99	时段	S99	时段	Z99	时段	S9	时段	V99
2014-2015历史表		2014-2015历史表		2014-2015历史表		2014-2015历史表		2014-2015历史表	
最小交易额: 2014-01-02 00:00:00	0	最小交易额: 2014-01-02 00:00:00	0	最小交易额: 2014-01-02 00:00:00	0	最小交易额: 2014-01-02 00:00:00	0	最小交易额: 2014-01-02 00:00:00	0
最大交易额: 2014-12-30 00:00:00	480	最大交易额: 2014-12-30 00:00:00	480	最大交易额: 2014-12-30 00:00:00	480	最大交易额: 2014-12-30 00:00:00	480	最大交易额: 2014-12-30 00:00:00	480
平均交易额: 116	116	平均交易额: 117	117	平均交易额: 145	145	平均交易额: 170	170	平均交易额: 119	119
可调度交易额: 127	127	可调度交易额: 142	142	可调度交易额: 137	137	可调度交易额: 109	109	可调度交易额: 133	133
总交易额: 10000000	10000000	总交易额: 9,546,705.0	9,546,705.0	总交易额: 14,413,343.52	14,413,343.52	总交易额: 16,305,760.0	16,305,760.0	总交易额: 11,209,300.0	11,209,300.0
增长率: 0.366, 976.5	0.366, 976.5	增长率: -0.17, 528	-0.17, 528	增长率: 49, 128	49, 128	增长率: 49, 298	49, 298	增长率: 0.098	0.098
平均增长率: -0.27%	-0.27%	平均增长率: -0.17%	-0.17%	平均增长率: 20, 328	20, 328	平均增长率: 27, 78	27, 78	平均增长率: 11, 668	11, 668
最大增长率: 175.423, 122.27	175.423, 122.27	最大增长率: 4, 194, 765.0	4, 194, 765.0	最大增长率: 4, 433, 343.52	4, 433, 343.52	最大增长率: 6, 305, 760.0	6, 305, 760.0	最大增长率: 2, 209, 300.0	2, 209, 300.0
最小增长率: -4, 212, 122.27	-4, 212, 122.27	最小增长率: -0.194, 430.6	-0.194, 430.6	最小增长率: -0.562, 642.56	-0.562, 642.56	最小增长率: -0.562, 642.56	-0.562, 642.56	最小增长率: -0.567, 231.14	-0.567, 231.14
可调度交易额: -66, 008	-66, 008	可调度交易额: -347, 304	-347, 304	可调度交易额: -467, 428	-467, 428	可调度交易额: -478, 876	-478, 876	可调度交易额: 0	0
总交易额: 0	0	总交易额: 0	0	总交易额: 0	0	总交易额: 0	0	总交易额: 0	0
日均交易额: 92.0	92.0	日均交易额: 55.0	55.0	日均交易额: 65.0	65.0	日均交易额: 52.0	52.0	日均交易额: 47.0	47.0
日均增长率: -0.562, 1	-0.562, 1	日均增长率: 9, 542.03	9, 542.03	日均增长率: 5, 043.74	5, 043.74	日均增长率: 11, 206	11, 206	日均增长率: 477.87	477.87
日均可调度交易额: 0.0	0.0	日均可调度交易额: 0.0	0.0	日均可调度交易额: 0.0	0.0	日均可调度交易额: 0.0	0.0	日均可调度交易额: 0.0	0.0
日均增长率: 0.0	0.0	日均增长率: 0.0	0.0	日均增长率: 0.0	0.0	日均增长率: 0.0	0.0	日均增长率: 0.0	0.0
日均可调度增长率: 0.0	0.0	日均可调度增长率: 0.0	0.0	日均可调度增长率: 0.0	0.0	日均可调度增长率: 0.0	0.0	日均可调度增长率: 0.0	0.0
日均增长率: -0.20%	-0.20%	日均增长率: -0.20%	-0.20%	日均增长率: 0.08%	0.08%	日均增长率: 0.12%	0.12%	日均增长率: 0.008	0.008
日均可调度增长率: 1.61%	1.61%	日均可调度增长率: 1.9%	1.9%	日均可调度增长率: 1.56%	1.56%	日均可调度增长率: 1.43%	1.43%	日均可调度增长率: 2.20%	2.20%
Charge Ratio: -0.25	-0.25	Charge Ratio: -0.74	-0.74	Charge Ratio: 0.6	0.6	Charge Ratio: 0.25	0.25	Charge Ratio: 0.32	0.32

图 6-11 全品种 2014—2015 年历史回测 (续一)

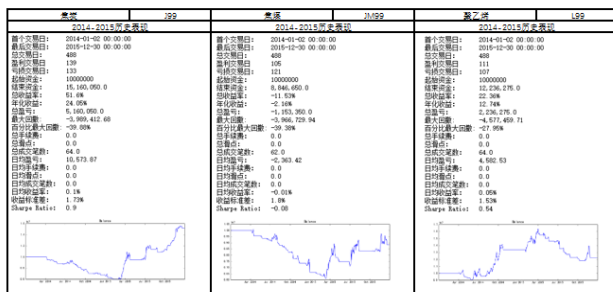


图 6-11 全品种 2014—2015 年历史回测（续二）

根据夏普比率 $>0.6$ 的准则，筛选出了 17 个品种，其历史表现和 2016 年预测表现如图 6-12 所示。投资组合在 2014—2015 年年化收益为 74.06%，百分比最大回撤为 -45.09%，夏普比率为 1.41。整体上还可以，但是 2016 年预测表现非常糟糕。这说明投资组合里噪声过多，不够稳健。

**筛选品种**

IF99,300,0,2,0,0,0  
 AL99,5,5,0,0,0  
 CU99,5,10,0,0,0  
 R899,10,1,0,0,0  
 AG99,15,1,0,0,0  
 WR99,10,1,0,0,0  
 ZN99,5,5,0,0,0  
 TA99,5,2,0,0,0  
 WH99,10,1,0,0,0  
 CF99,5,5,0,0,0  
 RM99,10,1,0,0,0  
 C99,10,1,0,0,0  
 JD99,5,1,0,0,0  
 I99,100,0,5,0,0,0  
 J99,100,0,5,0,0,0  
 B99,10,1,0,0,0  
 M99,10,1,0,0,0

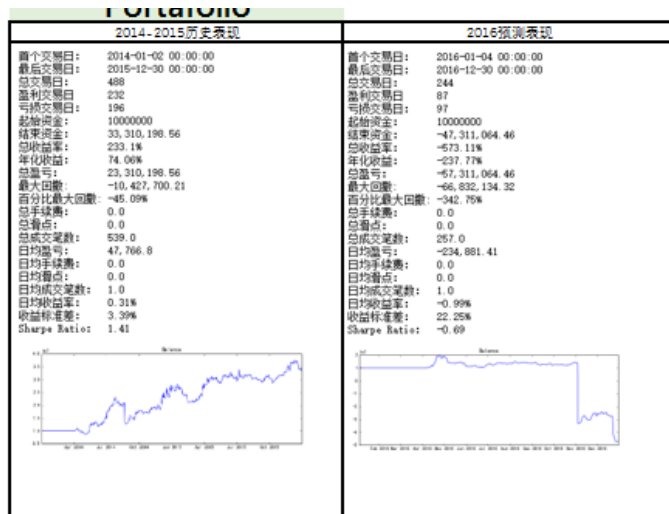


图 6-12 投资组合 2014—2015 年历史表现和 2016 年预测效果

下面分析一下挑选出来的品种成分，按交易所分类如下所示。



- 中金所：沪深 300 股指。
- 上期所：铝、铜、螺纹钢、白银、线材、锌。
- 郑商所：普麦、一号棉花、菜籽粕、PTA。
- 大商所：玉米、鸡蛋、铁矿石、焦煤、黄大豆 2 号、豆粕。

可以发现，占绝大部分的是成交量巨大的品种，仅仅是少部分上流动性差品种，如黄大豆 2 号。推测可能是这些流动性差的品种导致其投资组合在 2016 年亏损，故有必要进行滚动回测来剔除表现不好的品种。

### (2) 2015—2016 年测试

第一次从 35 个样本中筛选出 17 个，本次将继续剔除噪声因子，试图提高海龟组合的预测效果，测试如图 6-13 所示。

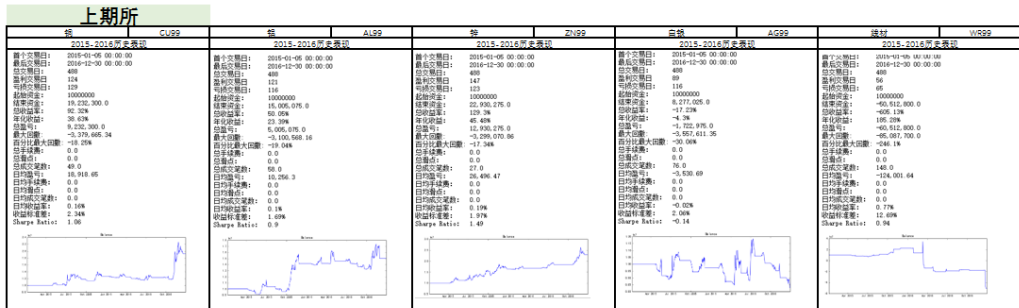
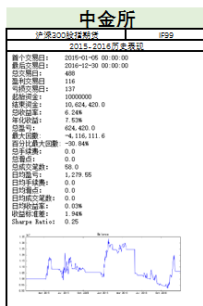


图 6-13 样本 2015—2016 年历史回测







图 6-15 样本 2015—2016 年历史回测

第三轮筛选后,得到由铝、铜、锌、普麦、铁矿石、焦炭、豆粕组成的海龟组合,都符合流动性强的特征,该组合的历史回测表现、2018 年预测效果,以及全时间区间表现如图 6-16 所示。

结果同样是不理想，在样本内表现出色，样本外却亏损。

夏普>0.6  
筛选品种  
AL99.5.0.0.0  
CU99.5.0.0.0  
ZIN99.5.0.0.0  
VW99.10.1.0.0.0  
I99.100.0.5.0.0.0  
J99.100.0.5.0.0.0  
M99.10.1.0.0.0

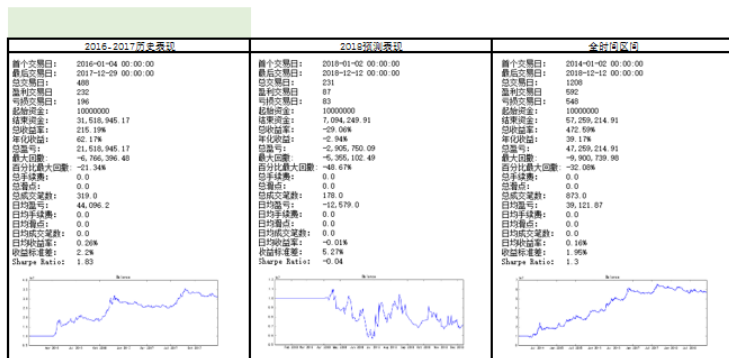


图 6-16 投资组合 2016—2017 年历史回测表现、2018 年预测效果，以及全时间区间表现

## 6.4.3 构建海龟组合的难点

在追求策略稳健性的前提下，通过传统的滚动回测并不能筛选出赢利组合，即这种适用于日内 CTA 策略的检验方法对海龟组合是无效的。

构建海龟组合的难点表现为以下几点。

### 1. 历史数据过少

从 2014 年到 2018 年这 5 年间，其日线数据仅仅有 1 200 条。其数据过少导致调整回望周期的灵活性降低，即最多回望周期只能设置为 2 年、3 年、4 年；同时数据少很难检验出策略的稳健性和有效性（原版海龟策略历史回测周期是 10 年）。

反观传统日内 CTA 策略，该策略是基于分钟级别数据的，1 年的分钟 K 线就接近 6 000 条。用传统的筛选品种的方法，2 年回望周期 K 线达到 12 000 条，在数据足够大的情况下更易于体现出策略的稳健性。

### 2. 回撤过大

海龟策略属于高风险、高收益的中低频策略，夸张的比喻就是“3 年不开张，开张吃 3 年”，在设定回测区间可能表现出持续的亏损，但也有可能之后在出现一波大行情，短期的赢利能够快速覆盖长期的亏损。但是滚动回测时因为夏普比

率过低而剔除该品种，可能会错过隐含品种的收益。

### 3. 筛选指标不稳健

本次滚动回测筛选标准是夏普比率，但是夏普比率对于策略回测时间区间起始点和终点非常敏感，对于趋势的把握不足。举例说明：在 2 年时间区间夏普比率是 1.3，若整体回测区间向后滚动 3 个月，则得到的夏普比率可能是 0.9。

故夏普比率不适应于海龟策略这种波动性大的类型，必须找到一种指标可以预测其趋势，比如“新式夏普比率”，在 2 年间比值是 1.3，整体向前或者向后滚动 3 个月，其比值也接近 1.3。

### 4. 样品数量噪声过大

尽管通过历史行情，2014 年前上市品种，已经把检测样本从 51 个降低到 35 个。但是对巨大的样品数量检验也造成很大的操作负担。例如，一些低流动性或者持续走震荡行数的品种，没有必要进行海龟策略的测试。

## 6.4.4 海龟组合筛选的解决方案

### 1. 重新进行初步筛选

重新对“流动性强”进行定义，在统计意义上，可以指行情波动巨大和具有大趋势，故通过 2 个指标专门对 35 个品种重新进行筛选。

#### (1) 调整后波动率比值

计算公式是收盘价的标准差/（合约最小价格变动 $\times 100$ ），调整后的波动率比值越大，代表其品种的相对波动幅度越大，趋势跟踪策略的赢利能力越强。

举个例子说明，假设趋势跟踪策略正确预测行情的概率是  $x\%$ ，其固定交易成本为  $c$ ，那么收益率  $r = (x\% \times \text{波动幅度}) - \text{固定成本}$ 。所以当波动幅度足够大时，交易才有利可图，否则交易赢利覆盖不了交易成本，导致单子做得越多，亏得也越多。

## (2) ADF 值

ADF 值是推论统计学的概念。通过 ADF 检验来得出其 ADF 值，然后与 10% 进行比较，若 ADF 值大于 10%，则证明完全不能拒绝原假设，即原假设成立。原假设如有单位根则代表着品种自相关性强，具有趋势行情。

总结一下，增加了两个初步筛选的标准，分别是：调整后波动率比值>1；ADF 值>10%。

代码实现如下。

```
import Numpy as np
import pandas as pd
import statsmodels.tsa.stattools as ts

def volPriceTick (SYM, PT):
    c=pymongo.MongoClient()
    symbol = SYM          #合约代号
    col = c['VnTrader_Daily_Db'][symbol]
    cx = col.find()
    l = list(cx)
    d = {}
    for key in l[0].keys():
        d[key] = []
    for data in l:
        for k, v in data.items():
            d[k].append(v)
    df = pd.DataFrame.from_dict(d)
    PriceTick = PT        #最小价格变动
    close = df['close']
    var = np.std(close)    #收盘价的标准差

    ratio = var / (PriceTick*100)    # 定义调整后波动率比值
    adfresult = ts.adfuller(close)   # 得到 ADF 值
    return ratio, adfresult
```

调用定义好的函数进行测试，得出调整后的波动率比率和 ADF 值如图 6-17 所示，品种是沪深 300 股指期货（IF99），其最小价格变动是 0.02。



```
In [16]: volPriceTick ( 'IF99' , 0.02 )
Out[16]: (342.1932441910671,
          (-1.8339745760731299,
           0.36372942729622326,
           24,
           1421,
           {'1%': -3.4349602407782758,
            '10%': -2.5678540089914974,
            '5%': -2.8635761009296763},
           15734.098959991614))
```

图 6-17 调整后的波动率比率和 ADF 值

调整后的波动率比值为 342.19，比值远大于 1，说明品种价格变化较为活跃；ADF 值是 0.36，同样大于 10%，说明其自相关性强，适用于趋势跟踪类型策略。

## 2. 回测时使用新的指标

夏普比率因其不稳健的特点，导致很难去识别一些行情趋势，而且对于回撤承受度不高，难以应用于像海龟策略这种高风险、高收益的策略。所以需要新的筛选指标。新的筛选指标应该具有 3 方面的特点。

- 稳健性强。
- 能够判断出整体趋势。
- 对回撤容忍度高。

回归夏普比率作为新的指标用于品种筛选，步骤如下。

(1) 用最小二乘法对累计资金曲线求得线性回归方程，得到斜率 `slope` 和截距 `intercept`。斜率为平均每天的盈亏资金，截距为回归起始资金（注意，仅仅是数学概念，可以大于或者小于 1 千万元，资金曲线表现得特别好的时候截距可以为负数）。通过斜率、截距与时间求得期末的回归资金 `RegendBalance`。

(2) 通过期末回归资金和截距的比值得到回归年化收益 `RegtotalReturn`。

(3) 把单利转换为复利，得到回归年化收益。

(4) 直接除以年化系数 `annualDays`（240 天），转化为回归日收益。



(5) 在计算夏普比率公式中, 把标准的日收益换成回归日收益, 得到回归夏普比率 (注意, 由于回归线是笔直的, 无标准差, 故以原始标准差计算)。

在海龟策略回测引擎的 `calculateResult` 中加入上面逻辑即可, 其代码如下:

```
#计算回归总收益
daysList = range(1,totalDays+1)

#slope , intercept = np.polyfit (changduList , balanceList , 1) #Numpy 方法
slope, intercept, r_value, p_value, slope_std_error =
stats.linregress(daysList , balanceList) #scipy 方法

RegendBalance = intercept + totalDays * slope
RegtotalReturn = (RegendBalance / intercept - 1) * 100

#计算回归年化收益
if RegtotalReturn > 0:
    RegannualizedReturn = (((1+RegtotalReturn/100)
**(annualDays/totalDays) )-1)*100
else:
    RegannualizedReturn = -(((1-RegtotalReturn/100)
**(annualDays/totalDays) )-1)*100

#计算回归日收益
RegdailyReturn = RegannualizedReturn/annualDays

#计算回归夏普比率
if returnStd:
    sharpeRatio = dailyReturn / returnStd * np.sqrt(annualDays)
    RegsharpeRatio = RegdailyReturn / returnStd * np.sqrt(annualDays)
else:
    sharpeRatio = 0
    RegsharpeRatio = 0
```

新指标的缺点:

- 高估资金曲线表现非常好的品种: 资金曲线非常好, 其回归线的截距 (即回归起始资金) 可能远远小于 1 千万元, 其斜率变大, 故标准夏普率可能是 1.3, 回归夏普比率可以达到 3.3。
- 低估资金曲线表现非常差的品种: 资金曲线在某个时间段剧烈向下移动, 其



回归线的截取可能远远大于 1 千万元，其斜率变小，故标准夏普比率可能是 -3，其回归夏普比率可以达 -0.6。

综合回归夏普比率的优势和缺点，一方面，我们得知其对回撤的容忍度高，可以摒弃噪声从而捕捉到趋势，故可以作为一个筛选标准；另一方面，有时候会出现高估或者低估行情的情况，故不能作为判断业绩表现的指标（如标准的夏普比率）。

### 3. 多种回望周期测试

把回望周期分别设置为 2 年、3 年和 4 年。理论上较短回测周期使用较为宽松的筛选标准（如回归夏普比率大于 0.4），较长回测周期使用较为苛刻的筛选标准（如回归夏普比率大于 0.8）。

当然为了测试的全面，通过 3 个不同的回望周期（如 2 年、3 年、4 年），以及不同的回归夏普比率（如 0.4、0.6、0.8）进行筛选，力求新的海龟组合能够较好地预测行情，表现出较高的稳健性。

## 6.4.5 重新构建投资组合

### 1. 初步筛选

除基于历史行情外，初步筛选还多加了品种波动率和自相关性，总结初步筛选条件为三点。

- 历史行情：2014 年 1 月 1 日前上市。
- 调整后波动率比值 > 1。
- ADF 值 > 10%。

根据初步筛选标准，剔除不符合要求的品种后，测试样本从调整前的 35 个缩小至 27 个，根据其调整后波动率比值的大小按从大到小排序，初步筛选品种结果如图 6-18 所示，其中，填充部分为成交量高的品种。

编号	品种	PriceTick	波动率比值	ADF值	成交量高品种
1	IF	0.2	34.21	36%	IF
2	J	0.5	10.09	58%	RB
3	ZN	5	8.06	76%	ZN
4	RB	1	7.18	53%	CU
5	CU	10	6.13	23%	AL
6	WR	1	6.07	39%	RU
7	TA	2	5.98	19%	SR
8	JM	0.5	5.42	41%	TA
9	RS	1	5.12	50%	RM
10	CF	5	5.08	39%	J
11	PB	5	5.007	67%	M
12	BB	0.05	4.53	71%	I
13	JD	1	4.47	38%	JM
14	B	1	4.26	17%	
15	BU	2	4.15	35%	
16	A	1	4.05	55%	
17	Y	2	3.72	48%	
18	C	1	3.49	63%	
19	SR	2	3.37	69%	
20	FB	0.05	3.16	40%	
21	M	1	3.1	23%	
22	AL	5	2.6	38%	
23	L	5	2	22%	
24	RI	1	1.78	18%	
25	V	5	1.39	15%	
26	WH	1	1.23	12%	
27	FG	1	1.09	52%	

图 6-18 初步筛选品种结果

一般来说，成交量高的品种，其波动率高，自相关性强，故具有正相关性。

根据交易所分类，这 27 个品种可划分成 4 部分。

- 中金所：IF。
- 上交所：ZN、RB、CU、WR、PB、BU、AL。
- 郑商所：TA、CF、RS、SR、RI、WH、FG。
- 大商所：J、BB、B、JM、JD、A、Y、C、FB、M、L、V。

## 2. 2 年回望周期测试

选择标准为回归夏普比率 $>0.4$ 。

(1) 2014—2015 年回测。

对初步筛选出来的样本进行 2014—2015 年历史回测，选择回归夏普比率 $>0.4$ 的品种，然后构成组合，如图 6-19 所示。



中 金 所	
伊藤2020年4月24日 星期四	IF99
2024.04.24 周五 收盘	
最新行情日期:	2024-04-24 15:00:00
最新行情时间:	10:00:00 - 10:00:00
最新行情品种:	400
最新行情代码:	154
最新行情名称:	137
标的物描述:	10000000
合约乘数:	10000000.000
合约单位:	144.599%
年化收益率:	-0.421%
盈亏:	14,460,000.000
最大盈亏:	43,125,748.0
当日最大盈亏:	-1.71%
当日涨跌幅:	0.0
当日最高价:	0.0
当日最低价:	0.0
当日开盘价:	0.0
当日收盘价:	0.0
当日成交量:	0.11
当日成交额:	0.26
当日持仓量:	1.21
当日持仓成本:	254.26%
当日持仓收益:	0.0
当日持仓成本率:	3.26

[illegible][illegible]

特	2014-2015年度表決	2015年
議決権行使率	2014-01-02 00:00:00	2015-01-02 00:00:00
議決権行使率	400	400
議決権行使率	100	100
議決権行使率	144	
議決権行使率	1000000	
議決権行使率	20,576,500.0	
議決権行使率	100.00%	
議決権行使率	42.19%	
議決権行使率	100.00%	
議決権行使率	-6,717,111.33	
議決権行使率	20.11%	
議決権行使率	0.0	
議決権行使率	0.0	
議決権行使率	48.0	
議決権行使率	21,089.22	
議決権行使率	0.0	
議決権行使率	0.1	
議決権行使率	0.17%	
議決権行使率	2.32%	
議決権行使率	1.14	
議決権行使率	80.00%	
議決権行使率	37.19%	
議決権行使率	1.43	

項目	2014-01-02 历史数据	05/09
最新开盘价	2014-01-02 00:00:00	
最新收盘价	2014-01-30 00:00:00	
最高价	480	
最低价	63	
中间价	64	
成交量	1000000	
成交金额	-0.004,090.0	
持仓量	-0.00	
持仓成本	495.954	
持仓均价	-12,004,290.0	
持仓成本差	-21,961,321.71	
持仓成本差率	-33.79%	
持仓成本差率	0.0	
持仓成本差率	0.0	
持仓成本差率	0.0	
持仓成本差率	-64,162.89	
持仓成本差率	0.00	
持仓成本差率	0.00	
持仓成本差率	1.98	
持仓成本差率	1.98	
持仓成本差率	0.71	
持仓成本差率	-0.71	
持仓成本差率	-0.006	
持仓成本差率	-0.14	

匯票	2014-01-09 星期四	01/09
1 香港匯票	2014-01-09 00:00:00	
2 倫敦匯票	2014-01-09 00:00:00	
3 紐約匯票	4880	
4 舊金山匯票	1256	
5 紐約匯票	125	
6 舊金山匯票	10000000	
7 倫敦匯票	12,434,400.0	
8 舊金山匯票	29.798	
9 紐約匯票	24.268	
10 倫敦匯票	2,434,400.0	
11 舊金山匯票	-0.372,193.6	
12 紐約匯票	-0.808	
13 倫敦匯票	0.0	
14 紐約匯票	60.0	
15 倫敦匯票	5,308.93	
16 紐約匯票	0.0	
17 倫敦匯票	0.12	
18 紐約匯票	0.18	
19 倫敦匯票	0.12	
20 紐約匯票	0.5	
21 倫敦匯票	30.198	
22 紐約匯票	13.898	
23 倫敦匯票	0.29	

**2014-2015赛季常规赛**

赛季常规赛日期:	2014-01-02 00:00:00
赛季常规赛日期:	2015-12-30 00:00:00
常规赛场次:	400
常规赛胜场:	159
常规赛负场:	120
常规赛平局:	10000000
常规赛进球:	11,205,250.0
常规赛失球:	12,096
平均进球数:	10.718
场均进球:	1,005,250.0
场均失球:	-9,750,718.81
进球比负场比例:	-20.75%
进球比胜场:	0
进球比平局:	0
进球比负场:	63.0
进球比平局:	2,469.77
进球比负场:	0.00
进球比平局:	0.00
进球比负场:	0.13
进球比平局:	0.00%
进球比负场:	2.16
进球比平局:	0.33
进球比负场:	16.49%
进球比平局:	0.51

帳號	日期
2014-2015 年度資料	
第 1 次月費	2014-01-01 至 2014-01-30
第 2 次月費	2014-02-01 至 2014-02-28
三個月	4000
第 1 次月費	91
第 2 次月費	90
第 3 次月費	90
第 4 次月費	15,629,240.0
第 5 次月費	66,296
第 6 次月費	24,296
第 7 次月費	6,426,360.0
第 8 次月費	-4,032,590.93
第 9 次月費	-21,476
第 10 次月費	0
第 11 次月費	46.0
第 12 次月費	11,439.53
第 13 次月費	0
第 14 次月費	0
第 15 次月費	0
第 16 次月費	0.19
第 17 次月費	1.49
第 18 次月費	3,643.61
第 19 次月費	49,629
第 20 次月費	1.12
第 21 次月費	1.37

[illegible]

2014 - 2015 历史表现

最新市盈率	2015-01-02 历史最高
最新市净率	2015-12-30 68.00 0.00
最新股息率	2015-12-30 1.20
最新市销率	145
最新市现率	10000000
最新市债率	5,026,160.0
净资产收益率	-29.16%
每股收益	-14.12
每股净资产	-27,290.00
每股净利润	-5,266.00, 0.1
每股现金流量	-43.12%
每股派息	0.0
每股股息	0.0
每股净资产	0.0
每股净利润	0.0
每股现金流量	-6,020.92
每股派息	0.0
每股股息	0.0
每股净资产	0.16
每股净利润	0.79
每股现金流量	-6.59
每股派息	0.00
每股股息	0.00
每股净资产	-31.78%
每股净利润	-15.96%
每股现金流量	-15.96%

摘要		VM492
2014.12.13 拜五 元旦		
總工作時間:	2014-01-05 08:00:00	
最後工作時間:	2015-12-30 00:00:00	
工作時間:	488	
每日工作時間:	119	
每日工作時間1:	105	
估計完成日:	1000000	
估計完成日:	91,997,376.05	
估計結果:	99.97%	
估計結果:	25.72%	
估計結果:	5,997,376.05	
估計結果:	25,574,299.30	
估計結果:	-17.12%	
估計結果:	0.0	
估計結果:	0.0	
估計結果:	60.0	
估計結果:	11,109.5	
估計結果:	0.0	
估計結果:	0.0	
估計結果:	0.13	
估計結果:	1.18	
估計結果:	1.47%	
估計結果:	2.60%	
估計結果:	10.27%	
估計結果:	90.96%	
估計結果:	1.32	

[illegible]

—— 一 号 站 库		CF92
2024-2025 历史库区		
每个小时流量：	2024-01-02 00:00:00	
每个小时流量：	2025-12-30 00:00:00	
平均流量：	400	
最大流量：	133	
最小流量：	132	
总流量：	10000000	
平均流量：	13,146,900.01	
总流量：	33.47%	
平均流量：	25.56%	
总流量：	3,146,900.01	
平均流量：	-128.73%	
总流量：	0.0	
平均流量：	6.0	
总流量：	6,448.57	
平均流量：	0.0	
总流量：	0.0	
平均流量：	0.13	
总流量：	0.07%	
平均流量：	1.42%	
总流量：	0.72	
平均流量：	10.90%	
总流量：	5.19%	
平均流量：	0.24	

图 6-19 样本 2014—2015 年历史回测

2014-2015 年决算		2015	2014-2015 年决算		2015
累计欠发工资	2014-01-02 00:00:00	0.00	累计欠发工资	2014-01-02 00:00:00	0.00
累计应发工资	2014-12-31 00:00:00	480	累计应发工资	2014-12-31 00:00:00	480
工资总额		56	工资总额		103
基本工资		53	基本工资		103
绩效工资		3	绩效工资		0.000000
津贴补贴		0.000000	津贴补贴		4.795156
加班工资		-0.000000	加班工资		-0.000000
社会保险费		-1316.716	社会保险费		-97.717
住房公积金		-115.070.0	住房公积金		-5.206.400.0
工会经费		0.000000	工会经费		-0.000.000.0
住房公积金		-0.000.000	住房公积金		-0.000.000.0
其他收入		0.0	其他收入		0.0
其他支出		0.0	其他支出		0.0
工资总额		-56.36	工资总额		-1000.91
绩效工资		0.0	绩效工资		0.0
津贴补贴		0.0	津贴补贴		0.0
加班工资		0.0	加班工资		0.0
社会保险费		0.18	社会保险费		0.26
住房公积金		0.18	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26
其他支出		-0.36	其他支出		0.26
住房公积金		-0.36	住房公积金		0.26
其他收入		-0.36	其他收入		0.26

大商所

[illegible]

行情宝	2014-2015历史表现	P99	历史宝	6889	行情宝	2014-2015历史表现	P99	历史宝	99	行情宝	2014-2015历史表现	P99	历史宝	199
最大回撤:	2014-01-02 00:00:00		最大回撤:	2014-01-02 00:00:00		最大回撤:	2014-01-02 00:00:00		最大回撤:	2014-01-02 00:00:00		最大回撤:	2014-01-02 00:00:00	
最小回撤:	2015-12-30 00:00:00		最小回撤:	2015-12-30 00:00:00		最小回撤:	2015-12-30 00:00:00		最小回撤:	2015-12-30 00:00:00		最小回撤:	2015-12-30 00:00:00	
持仓量:	408		持仓量:	408		持仓量:	408		持仓量:	408		持仓量:	408	
持仓成本:	139		持仓成本:	139		持仓成本:	139		持仓成本:	139		持仓成本:	139	
持仓均价:	131		持仓均价:	147		持仓均价:	137		持仓均价:	139		持仓均价:	131	
持仓盈亏:	-5,521,500.0		持仓盈亏:	5,846,200.0		持仓盈亏:	10,000,000.0		持仓盈亏:	16,820,760.0		持仓盈亏:	12,000.0	
年化收益:	-19.31%		年化收益:	-41.55%		年化收益:	14,413.34%		年化收益:	1.29%		年化收益:	11,608.220.0	
年化波动:	300.0%		年化波动:	-21.9%		年化波动:	20.92%		年化波动:	27.7%		年化波动:	12.00%	
盈亏:	-13,531,500.0		盈亏:	-6,156,300.0		盈亏:	4,413,343.52		盈亏:	1,290.0		盈亏:	1,200.00%	
持仓盈亏:	-14,161,300.0		持仓盈亏:	-6,156,400.0		持仓盈亏:	4,413,343.52		持仓盈亏:	-5,502,902.22		持仓盈亏:	-5,902,370.54	
持仓均价:	-139.23%		持仓均价:	147.00%		持仓均价:	20.40%		持仓均价:	-14.42%		持仓均价:	-77.87%	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.14	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓盈亏:	-7,727.97		持仓盈亏:	-6,513.03		持仓盈亏:	9,403.74		持仓盈亏:	16,109.20		持仓盈亏:	2,437.07	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0	
持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0		持仓均价:	0.0	
持仓成本:	0.0		持仓成本:	0.0		持仓成本:	0.0		持仓成本					

基金	199	基金	1999	基金	199
2014-2015 历史表现		2014-2015 历史表现		2014-2015 历史表现	
最小回报率	2014-01-02 00:00:00	最小回报率	2014-01-02 00:00:00	最小回报率	2014-01-02 00:00:00
平均回报率	489	平均回报率	489	平均回报率	489
最大回报率	139	最大回报率	150	最大回报率	111
波动率	1139	波动率	1221	波动率	489
标准差	1000000	标准差	1000000	标准差	1000000
夏普比率	15.160, 050.0	夏普比率	5.846, 050.0	夏普比率	11.236, 178.0
回撤	0.0	回撤	0.0	回撤	0.0
回撤比率	51.48	回撤比率	-1.124	回撤比率	22.308
平均回撤	24.858	平均回撤	-18.202.0	平均回撤	7.448
回撤时间	5, 160, 050.0	回撤时间	5, 160, 050.0	回撤时间	2, 236, 178.0
最大回撤	-2, 990, 442.0	最大回撤	-9, 990, 729.94	最大回撤	-4, 517, 046.71
持仓成本	-20.894	持仓成本	-20.326	持仓成本	-27.954
持仓成本比率	0.0	持仓成本比率	0.0	持仓成本比率	0.0
持仓成本时间	0.0	持仓成本时间	0.0	持仓成本时间	0.0
持仓成本比率	0.0	持仓成本比率	0.0	持仓成本比率	0.0
持仓成本时间	10, 673.97	持仓成本时间	-2, 363.42	持仓成本时间	4, 905.13
持仓成本比率	0.0	持仓成本比率	0.0	持仓成本比率	0.0
持仓成本时间	0.0	持仓成本时间	0.13	持仓成本时间	0.13
持仓成本比率	0.18	持仓成本比率	-0.576	持仓成本比率	0.054
持仓成本时间	0.0	持仓成本时间	1.96	持仓成本时间	0.0
持仓成本比率	0.126	持仓成本比率	-0.326	持仓成本比率	0.54
持仓成本时间	22.74	持仓成本时间	-1.16	持仓成本时间	19.376
持仓成本比率	0.03	持仓成本比率	0.02	持仓成本比率	0.02

图 6-19 样本 2014—2015 年历史回测 (续)

根据回归夏普比率 $>0.4$ 的准则，筛选出了 14 个品种，其历史表现和 2016 年预测表现如图 6-20 所示。投资组合在 2014—2015 年年化收益为 96.46%，百分比最大回撤为-32.75%，夏普比率达 2.04，资金曲线平滑且整体向上，但是 2016 年预测表现不佳，需要剔除更多噪声因子。

筛选品种

IF99,300,0,2,0,0,0  
AL99,5,5,0,0,0  
CU99,5,10,0,0,0  
RB99,10,1,0,0,0  
PB99,5,5,0,0,0  
ZN99,5,5,0,0,0  
TA99,5,2,0,0,0  
WH99,10,1,0,0,0  
C99,10,1,0,0,0  
I99,100,0,5,0,0,0  
J99,100,0,5,0,0,0  
B99,10,1,0,0,0  
M99,10,1,0,0,0  
L99,5,5,0,0,0

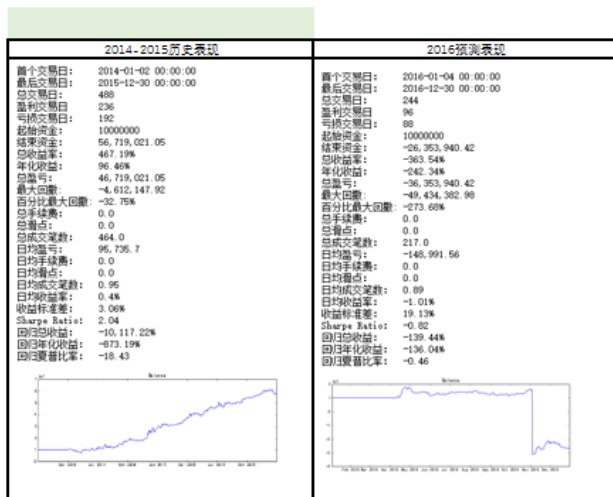


图 6-20 投资组合 2014—2015 年历史表现和 2016 年预测效果

下面分析一下挑选出来的品种成分，按交易所分类如下。

- 中金所：沪深 300 股指。
- 上期所：铝、铜、螺纹钢、铅、线材、锌。
- 郑商所：普麦、PTA。
- 大商所：玉米、铁矿石、焦煤、黄大豆 2 号、豆粕、聚乙烯。

## (2) 2015—2016 年回测。

同样对剩下的样本进行 2015—2016 年历史回测，选择回归夏普比率 $>0.4$ 的品种，然后构成组合，如图 6-21 所示。

[illegible][illegible][illegible][illegible]

图 6-21 样本 2015—2016 年历史回测

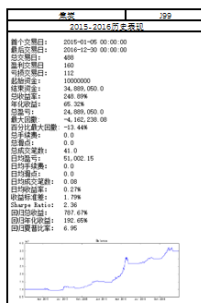


图 6-21 样本 2015—2016 年历史回测（续）

经过第二轮筛选后，剩下 9 个品种，同样按照交易所分类，如下所示。

- 上期所：铝、铜、螺纹钢、锌。
- 郑商所：普麦。
- 大商所：玉米、铁矿石、焦炭、豆粕。

在新的投资组合中，年化收益达 92.04%，百分比最大回撤是-16.8%，夏普比率达 2.4，整体资金曲线比较平滑。在 2017 年预测表现理想，年化收益为 46.49%，百分比最大回撤是-30.45%，夏普比率达 1.09，如图 6-22 所示。

#### 筛选品种

AL99.5.5.0.0.0  
CU99.5.10.0.0.0  
RS99.10.1.0.0.0  
ZN99.5.5.0.0.0  
WH99.10.1.0.0.0  
C99.10.1.0.0.0  
I99.100.0.5.0.0.0  
J99.100.0.5.0.0.0  
M99.10.1.0.0.0

2

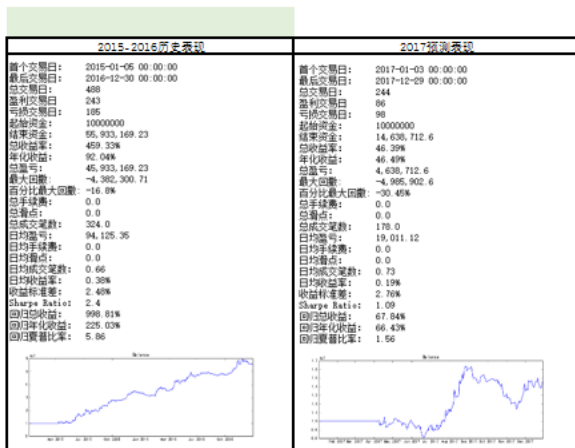


图 6-22 投资组合 2015—2016 年历史表现和 2017 年预测效果



(3) 2016—2017 年回测。

最后一轮策略，将挑选出最终的品种组成海龟组合，如图 6-23 所示。

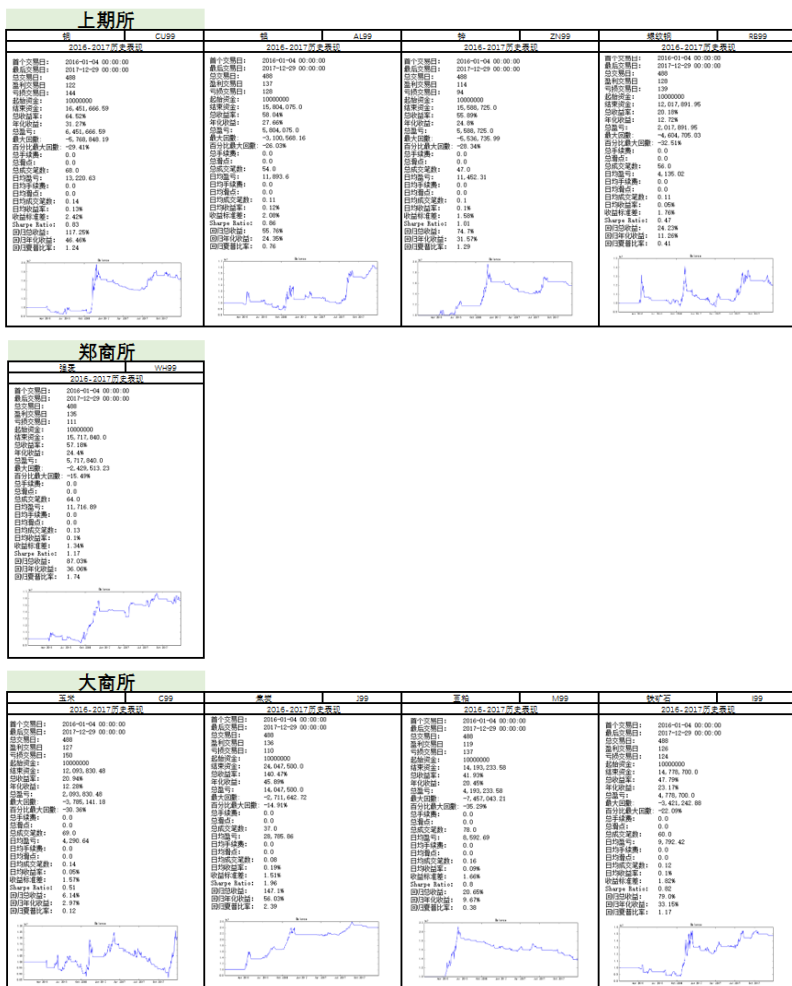


图 6-23 样本 2016—2017 年历史回测

第三轮筛选后，基于回归夏普比率 $>0.4$ 得到由铝、铜、锌、普麦、铁矿石、焦炭、螺纹钢组成的海龟组合，2016—2017 年标准夏普比率达 1.56，2018 年预测的夏普比率是 $-0.12$ ，全时间区间的夏普比率是 1.17，投资组合效果差强人意。如

图 6-24 所示。

夏普>0.4

筛选品种

AL99.5.0.0.0  
CU99.5.0.0.0  
RB99.10.1.0.0  
ZN99.5.0.0.0  
WH99.10.1.0.0  
I99.100.0.5.0.0  
J99.100.0.5.0.0

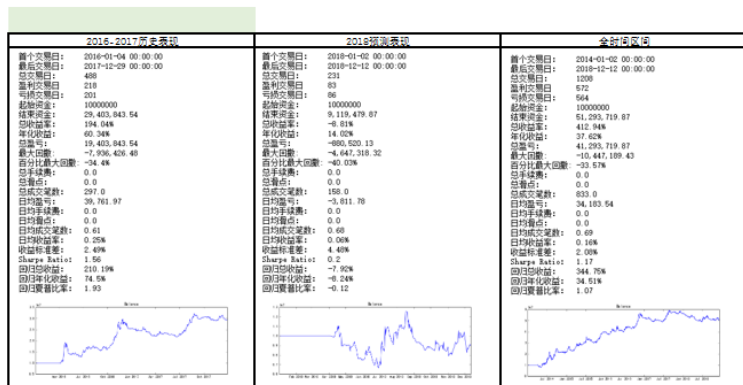


图 6-24 海龟组合（回归夏普比率>0.4）

基于上面 2 年回望周期所展示的回测图，可以更加便捷地更改筛选标准，不用重新进行测试即可得到结果，故下面把筛选标准改成回归夏普比率>0.6，其测试情况如图 6-25 所示。

夏普>0.6

筛选品种

AL99.5.0.0.0  
CU99.5.0.0.0  
ZN99.5.0.0.0  
I99.100.0.5.0.0  
J99.100.0.5.0.0

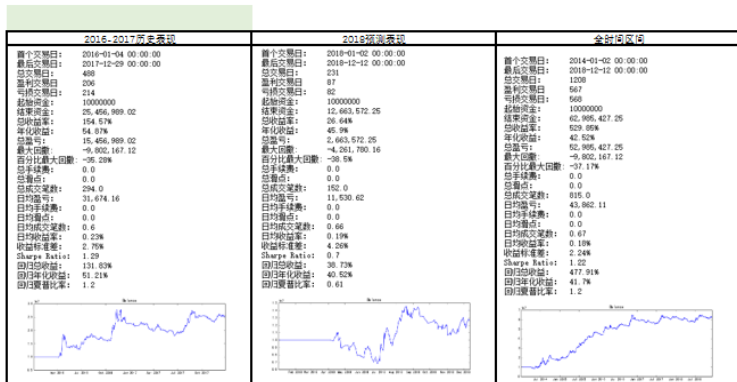


图 6-25 海龟组合（回归夏普比率>0.6）

把筛选标准提高到 0.6 后，得到的样本数量降低到 5 个，分别是铝、铜、锌、铁矿石、焦炭，2016—2017 年标准夏普比率达 1.29，2018 年预测的夏普比率是 0.7，全时间区间的夏普比率表现是 1.22。

若把筛选标准提升至回归夏普比率 $>0.8$ ，则得到 4 个样本品种：铜、锌、铁矿石、焦炭，2016—2017 年标准夏普比率达 1.87，2018 年预测的夏普比率是 $-0.03$ ，全时间区间的夏普比率表现是 1.38，如图 6-26 所示。

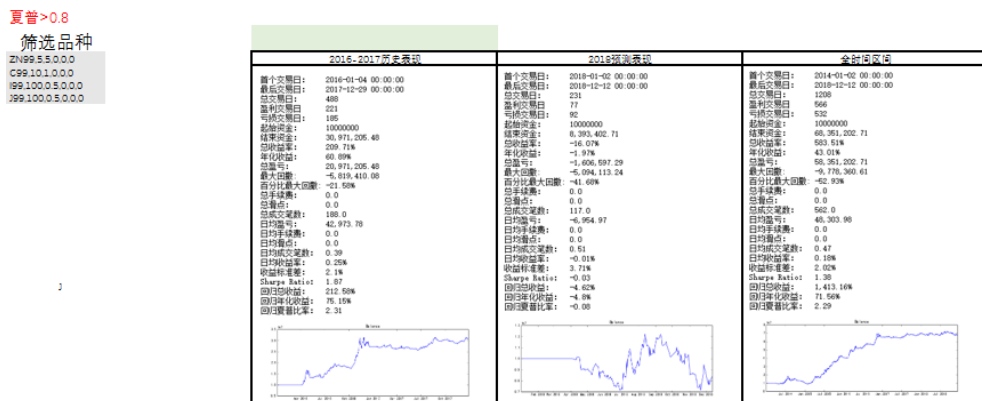


图 6-26 海龟组合（回归夏普比率 $>0.8$ ）

### 3. 3 年回望周期测试

选择标准为回归夏普比率大于 0.4。

(1) 2014—2016 年回测。

对初步筛选出来的样本进行 2014—2016 年回测，选择回归夏普比率大于 0.4 的品种，然后构成组合，如图 6-27 所示。



图 6-27 样本 2014—2016 年历史回测

[illegible]

图 6-27 样本 2014—2016 年历史回测 (续一)

图 6-27 样本 2014—2016 年历史回测 (续)

根据回归夏普比率 $>0.4$ 的准则，筛选出了 17 个品种，其历史表现和 2017 年预测表现如图 6-28 所示。投资组合在 2014—2016 年年化收益为 73.7%，百分比最大回撤为 -22.99%，夏普比率达 2.48，资金曲线平滑且整体向上，但是 2017 年预测表现不佳，资金曲线不断上下震荡，夏普比率仅仅是 -0.23。

筛选品种

IF99.300.0.2.0.0.0  
AL99.5.5.0.0.0  
CU99.5.10.0.0.0  
RB99.10.1.0.0.0  
PB99.5.5.0.0.0  
ZN99.5.5.0.0.0  
WH99.10.1.0.0.0  
CF99.5.5.0.0.0  
C99.10.1.0.0.0  
F99.100.0.5.0.0.0  
J99.100.0.5.0.0.0  
V99.5.5.0.0.0  
J99.5.1.0.0.0  
M99.10.1.0.0.0  
JH99.60.0.5.0.0.0

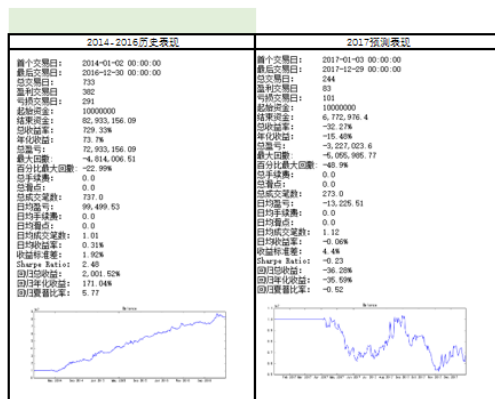


图 6-28 投资组合 2014—2016 年历史表现和 2017 年预测效果

## (2) 2015—2017 年回测。

2015—2017 年回测是最后一轮策略，选择回归夏普比率 $>0.4$ 的品种，然后构成最终的海龟组合，如图 6-29 所示。



图 6-29 样本 2015—2017 年历史回测



图 6-29 样本 2015—2017 年历史回测 (续)

根据回归夏普比率 $>0.4$ 的准则,筛选出 14 个品种,其历史表现和 2018 年预测表现如图 6-30 所示。投资组合在 2015—2017 年年化收益为 62.2%,百分比最大回撤-23.21%,夏普比率达 1.82;但是 2017 年震荡向上,夏普比率达 0.59,故 2018 年是赢利的,全时间区间的夏普比率为 1.42。





筛选品种

AL99.5.5.0.0.0  
CU99.5.10.0.0.0  
RB99.10.1.0.0.0  
R99.5.5.0.0.0  
ZN99.5.5.0.0.0  
WH99.10.1.0.0.0  
CF99.5.5.0.0.0  
C99.10.1.0.0.0  
I99.100.0.5.0.0.0  
I99.100.0.5.0.0.0  
M99.10.1.0.0.0  
JM99.50.0.5  
V99.5.5.0.0.0  
I999.5.1.0.0.0

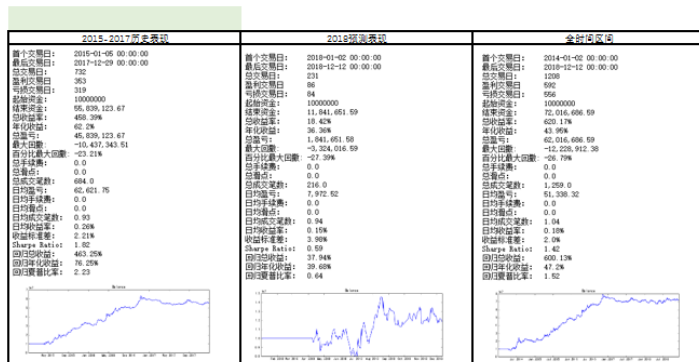


图 6-30 海龟组合 (回归夏普比率&gt;0.4)

若把筛选标准改成回归夏普比率>0.6 后, 投资组合的品种数量降低到 12 个, 则其历史表现和 2018 年预测表现如图 6-31 所示。投资组合在 2015—2017 年年化收益为 63.36%, 百分比最大回撤为-20.78%, 夏普比率达 1.81; 2018 年预测表现是先发生回撤然后行情走好, 夏普比率达 0.59; 全时间区间的夏普比率为 1.5。

夏普&gt;0.6

筛选品种

AL99.5.5.0.0.0  
CU99.5.10.0.0.0  
RB99.10.1.0.0.0  
R99.5.5.0.0.0  
ZN99.5.5.0.0.0  
WH99.10.1.0.0.0  
CF99.5.5.0.0.0  
C99.10.1.0.0.0  
I99.100.0.5.0.0.0  
I99.100.0.5.0.0.0  
M99.10.1.0.0.0  
JM99.50.0.5  
V99.5.5.0.0.0  
I999.5.1.0.0.0

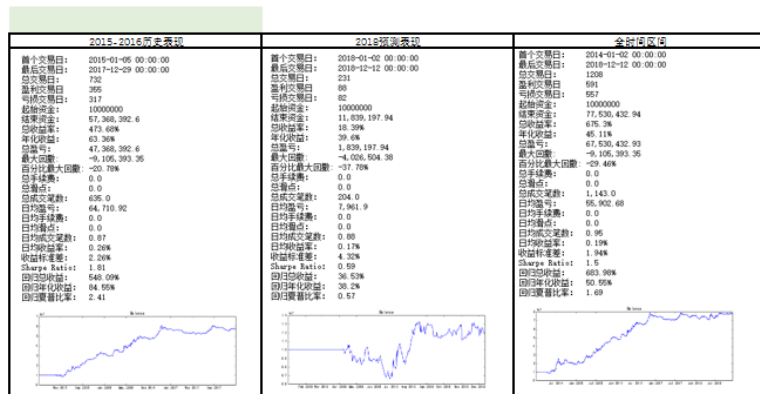


图 6-31 海龟组合 (回归夏普比率&gt;0.6)

若把筛选标准改成回归夏普比率>0.8 后, 投资组合的品种数量降低到 9 个, 则其历史表现和 2018 年预测表现如图 6-32 所示。投资组合在 2015—2017 年年化收益为 63.85%, 百分比最大回撤是-24.24%, 夏普比率达 1.84; 2018 年预测表现是: 先发生回撤然后行情走好, 夏普比率达 0.57; 全时间区间的夏普比率为 1.52。



夏普>0.8

筛选品种

AL99.5.0.0.0  
R99.10.1.0.0.0  
Z199.5.0.0.0  
V199.10.1.0.0.0  
C99.10.1.0.0.0  
I99.10.0.5.0.0.0  
J99.10.0.5.0.0.0  
M99.10.1.0.0.0  
JH99.50.0.5.0.0.0

2

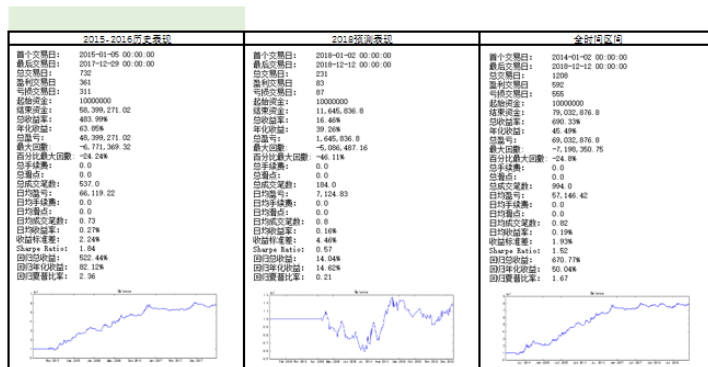


图 6-32 海龟组合（回归夏普比率>0.8）

## 4. 4 年回望周期测试

选择标准是回归夏普比率>0.4，对初步筛选出来的样本进行 2014—2017 年历史回测，然后一步构成组合，如图 6-33 所示。

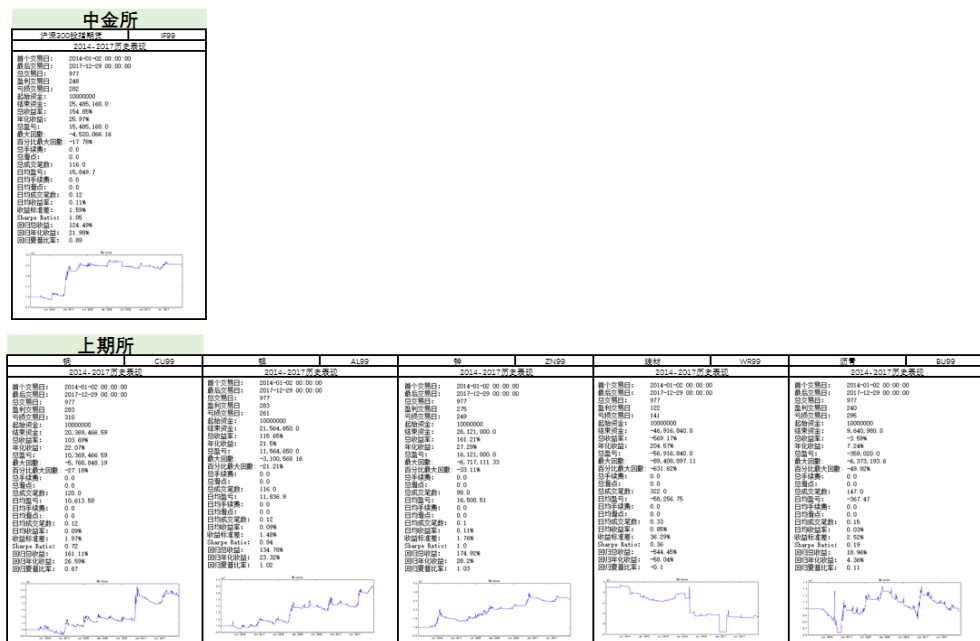


图 6-33 样本 2014—2017 年历史回测



[illegible]

郑商所

[illegible][illegible]

大商所

[illegible]

图 6-33 样本 2014—2017 年历史回测 (续一)



图 6-33 样本 2014—2017 年历史回测 (续二)

根据回归夏普比率 $>0.4$ 的准则, 筛选出 15 个品种, 其历史表现和 2017 年预测表现如图 6-34 所示。投资组合在 2015—2017 年年化收益为 57.2%, 百分比最大回撤为-22.61%, 夏普比率达 1.82; 但是 2018 年资金曲线上上下下震荡, 夏普比率仅为 0.38; 全时间区间的夏普比率为 1.61。

夏普 $>0.4$

筛选品种

IF99.300.0.0.0.0  
AL99.5.0.0.0  
CU99.5.10.0.0  
R99.10.1.0.0.0  
P99.5.5.0.0.0  
Z99.5.0.0.0  
W99.10.1.0.0.0  
C99.5.5.0.0.0  
S99.10.1.0.0.0  
M99.10.1.0.0.0  
U99.10.0.5.0.0.0  
S99.10.0.5.0.0.0  
V99.5.0.0.0  
J99.60.0.5.0.0.0  
J99.5.1.0.0.0

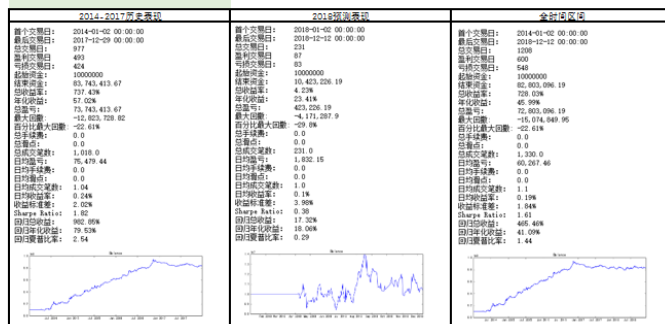


图 6-34 海龟组合 (回归夏普比率 $>0.4$ )

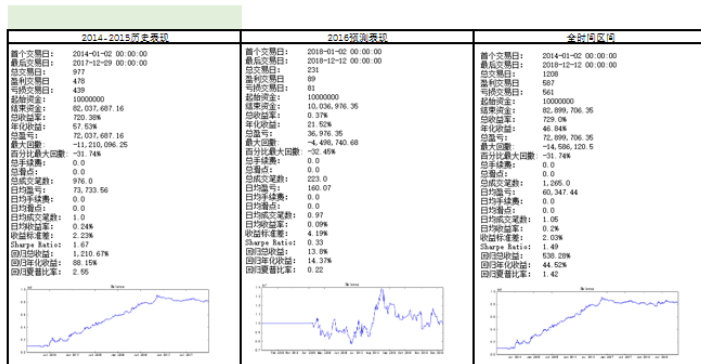


根据回归夏普比率 $>0.6$ 的准则，剔除了“鸡蛋”这个品种，其历史表现和 2017 年预测表现如图 6-35 所示。投资组合在 2015—2017 年年化收益为 57.2%，百分比最大回撤-22.61%，夏普比率达 1.67；但是 2018 年夏普比率为 0.33；全时间区间的夏普比率为 1.49。

夏普 $>0.6$ 

筛选品种

IF99.900.0.0.0.0  
AL99.5.0.0.0  
CU99.5.10.0.0.0  
RB99.10.1.0.0.0  
PB99.5.5.0.0.0  
ZN99.5.5.0.0.0  
WH99.10.1.0.0.0  
C99.5.5.0.0.0  
C99.10.1.0.0.0  
M99.10.1.0.0.0  
I99.100.5.0.0.0  
I99.100.5.0.0.0  
V99.5.5.0.0.0  
M99.60.5.0.0.0

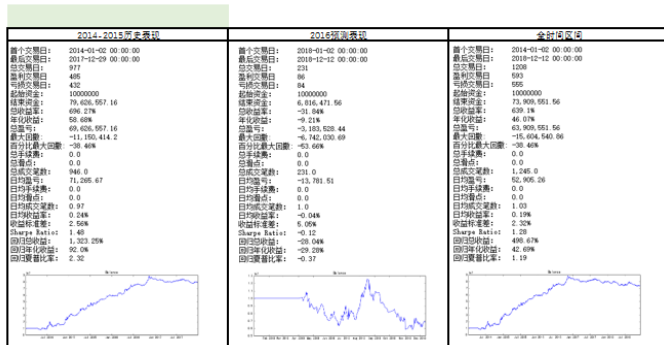
图 6-35 海龟组合（回归夏普比率 $>0.6$ ）

根据回归夏普比率 $>0.8$ 的准则，剔除“一号棉花”这个品种，其历史表现和 2017 年预测表现如图 6-36 所示。投资组合在 2015—2017 年年化收益为 58.68%，百分比最大回撤-38.46%，夏普比率达 1.48；但是 2018 年夏普比率为-0.12；全时间区间的夏普比率为 1.28。

夏普 $>0.8$ 

筛选品种

IF99.900.0.0.0.0  
AL99.5.0.0.0  
CU99.5.10.0.0.0  
RB99.10.1.0.0.0  
PB99.5.5.0.0.0  
ZN99.5.5.0.0.0  
WH99.10.1.0.0.0  
C99.5.5.0.0.0  
C99.10.1.0.0.0  
M99.10.1.0.0.0  
I99.100.5.0.0.0  
I99.100.5.0.0.0  
V99.5.5.0.0.0  
M99.60.5.0.0.0

图 6-36 海龟组合（回归夏普比率 $>0.8$ ）

根据回归夏普比率 $>1.0$ 的准则，剔除的品种包括沪深 300 股指、铜、铅，其历史表现和 2018 年预测表现如图 6-37 所示。投资组合在 2015—2017 年年化收益为 53.4%，百分比最大回撤为 -28.38%，夏普比率达 1.37；2018 年夏普比率是 0.43；全时间区间的夏普比率为 1.19。

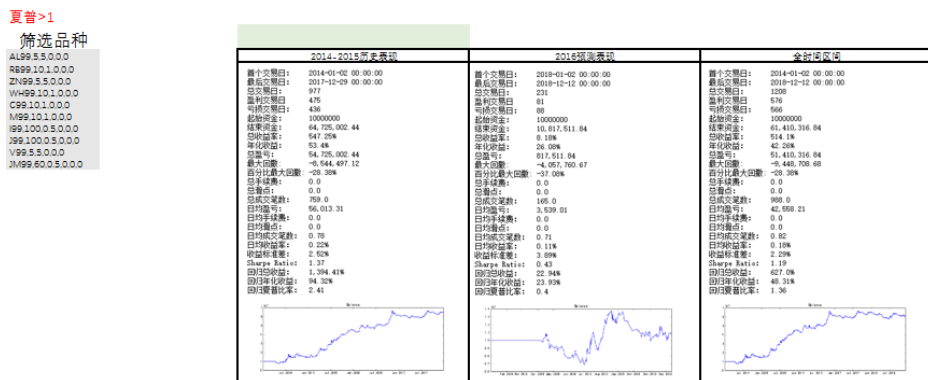


图 6-37 海龟组合 (回归夏普比率 $>1.0$ )

## 5. 多种回望周期回测总结

在上面的测试中，通过不同的回望周期（如 2 年、3 年、4 年）和不同的筛选标准得到 10 个备选的海龟组合。从图 6-38 可以看出，整体表现最好的是回望周期为 3 年，筛选标准是回归夏普比率 $>0.6$ 的组合，故该投资组合成为最终的海龟组合，用于其他关键要素的验证，比如单位头寸限制、长短周期入场信号、首次赢利过滤等。

Benchmark	回查窗口：2年						
	2018预测	全时间区间					
夏普比率>0.6	-0.04	1.3					
夏普比率>0.8	0.11	1.23					
改进	回查窗口：2年		回查窗口：2年		回查窗口：4年		
	2018预测	全时间区间	2018预测	全时间区间	2018预测	全时间区间	
	回归夏普比率>0.4	0.2	1.17	0.59	1.42	0.38	1.61
	回归夏普比率>0.6	0.7	1.22	0.59	1.5	0.33	1.49
	回归夏普比率>0.8	-0.03	1.38	0.57	1.52	-0.12	1.28
	回归夏普比率>1.0					0.43	1.11

以上数组指的是(标准)夏普比率

图 6-38 海龟组合备选表格

## 6.5 长短周期信号检验

原版海龟策略的出入场交易信号分为两个版本。

- **短周期版本。**①入场信号：若期货指数价格突破 20 日最高价/最低价，则买入/卖出 1 个头寸单位。过滤条件：上一次突破是赢利性突破，则当前入市信号无效（其保障性突破点为在 55 日通道入市）。②离场信号：采用 10 日唐奇安通道突破退出法则，即多头价格跌破 10 日最低点离场，空头价格超过 10 日最高点离场。
- **长周期版本。**①入场信号：若价格突破 55 日最高价/最低价，则买入/卖出 1 个头寸单位。对于长周期版本来说，所有突破都被视为有效信号，无论上一次突破是亏损还是赢利的。②离场信号：采用 20 日唐奇安通道突破退出法则，即多头价格跌破 20 日最低点离场，空头价格超过 10 日最高点离场。

《海龟交易法则》的作者费思认为这两个版本的出入场信号都是有效的，“海龟”们可以选择其中一个版本进行交易，也可以把资金分割成 2 部分，一部分使用短周期版本，另一部分使用长周期版本。

但是这都是 30 年前在美国市场有效的版本，照搬到当今的国内期货市场不能够保证其有效性，故需要对其进行检验。

检验方法是把海龟策略的信号系统拆分成 3 个：标准版本（长周期+短周期）、短周期版本和长周期版本。用组合进行验证，其效果如图 6-39 所示。

结果显示：标准版本年化收益为 45.11%，百分比最大回撤-29.46%，夏普比率达 1.5。长周期版本年化收益为 39.96%，百分比最大回撤为-25.81%，夏普比率达 1.44。短周期版本年化收益为 44.34%，百分比最大回撤为-25.61%，夏普比率达 1.65。因此可以推测出是长周期版本整体效果较差，拖累了短周期版本的效果，从而拉低整体的夏普比率。

### 投资组合

AL99.5,5.0,0.0  
CU99.5,10.0,0.0  
RB99.10,1.0,0.0  
ZN99.5,5.0,0.0  
WH99.10,1.0,0.0  
CF99.5,5.0,0.0  
C99.10,1.0,0.0  
J99.100,0.5,0.0,0.0  
M99.10,1.0,0.0  
V99.5,5.0,0.0  
JM99.60,0.5,0.0,0.0

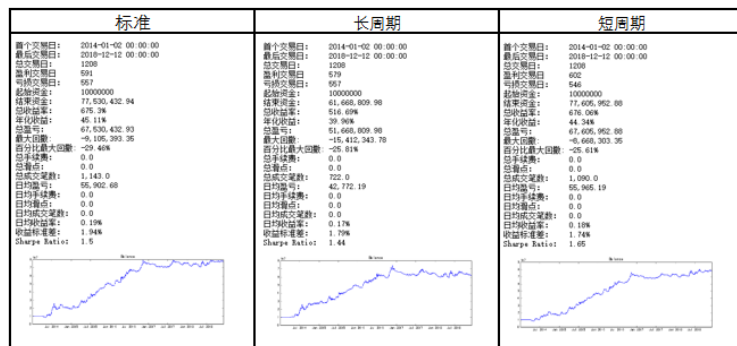


图 6-39 海龟组合长短周期信号检验

从海龟组合的角度来看，应该剔除长周期版本的出入场信号，仅仅保留短周期版本。为了再一次验证其结论的有效性，逐个对单品种进行测试，然后统计其 3 个版本的夏普比率。为了表述起来更加直观，本次使用打分法，步骤如下。

(1) 根据夏普比率的测试效果对 3 个版本的海龟策略进行排序。

(2) 对于不同排名附加权重不同，如排名第一得 1 分，排名第二得 0.8 分，排名第三得 0 分。

(3) 统计 3 个版本海龟策略的总分。

单品种如图 6-40 所示。中金所短周期版本总分最高，为 2.8 分，远高于标准版本（1 分）和长周期版本（0.8 分），反过来短周期版本得分最低。

总分=1\*排名第一+0.8\*排名第二

策略版本	IF	IC	IS	排名第一	排名第二	总分
标准	2	1	3	1	0	1
中金所	2	2	2	2	1	0.8
短周期	1	2	1	2	2	2.8

部门品种										金品种			
策略版本	SR	CV	AL	RV	SL	IR	SH	AW	AG	PR	WIS	IC	SC
标准	2	2	2	1	2	2	2	2	2	2	2	2	2
中金所	2	1	2	2	2	1	2	2	2	2	2	2	2
短周期	1	2	1	2	1	2	1	2	2	1	2	2	2

部门品种										金品种									
策略版本	TA	MA	AP	SS	PM	SF	SG	SM	SC	WH	CF	SR	CH	SI	WH	IR	LR	CV	IC
标准	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
中金所	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
短周期	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

部门品种										金品种									
策略版本	JM	LI	RI	C	A	IS	V	S	PR	SD	LV	IL	PP	CS	排名第一	排名第二	总分	排名第一	排名第二
标准	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
中金所	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
短周期	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

图 6-40 单品种长短周期分数表





同样的结论也应用于其他三个交易所的组合，如上期所长周期全品种组合得分最低（5分），热门品种组合也是最低（2.8分）；郑商所长周期的全品种组合为5分，热门品种组合为2.8分；大商所长周期的全品种组合为4.6分，热门品种组合为1分。

综上所述，国内四大交易所长周期在热门品种组合和全品种组合得分均最低，故判断长周期入场信号无效，应该剔除。新的海龟策略仅仅保留短周期信号，并且以后的验证也是基于短周期版本的海龟策略。

## 6.6 上一笔赢利过滤检验

上一笔赢利过滤的意思是，若上一次交易是赢利的，则当前的交易信号无效，即当前不进行交易。基于20日唐奇安通道，费思认为，若上一次突破点赢利，那么新突破点可能离当前价远，因为有可能是个55日突破点，若上一次突破点亏损，那么新突破点将更加接近当前价格。

从传统日内中高频CTA策略的视角看这是非常主观的解释，费思认为这几乎不可能连续两次出现大行情，但事实是基于分钟级别数据的CTA策略有可能出现两次大行情，故首次赢利过滤的作用仅仅是节省手续费和滑点，而错过了潜在的巨大收益。

故基于日线级别中低频CTA策略的首次赢利过滤是否有效还需要进行验证。本次测试的基准是剔除长周期版本的“新”海龟策略，海龟组合是在6.4节中挑选的投资组合（回望周期为3年；筛选标准是回归夏普比率 $>0.6$ ），然后把海龟策略分为含上一笔赢利过滤和不含上一笔赢利过滤两种，上一笔赢利过滤对比图如图6-41所示。



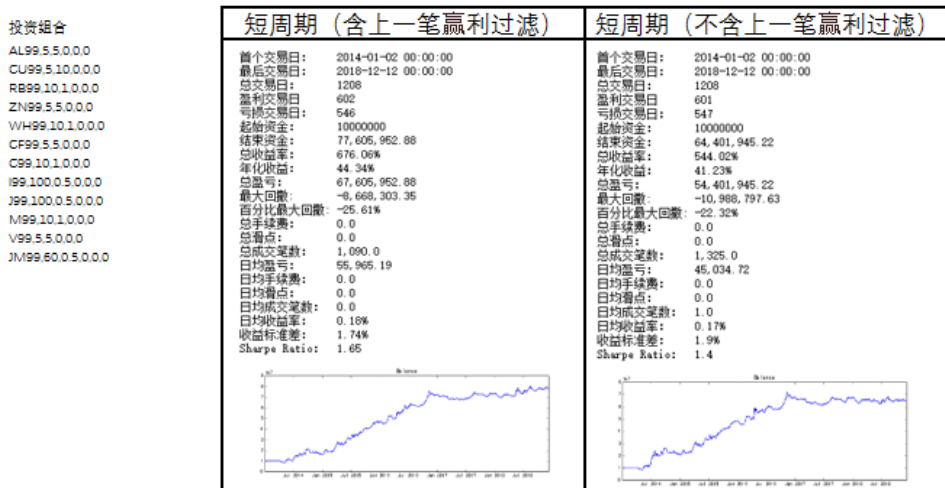


图 6-41 上一笔赢利过滤对比图

从图 6-41 中可以看出如下结论。

- 不含上一笔赢利过滤版本: 总成交笔数 1325, 年化收益为 41.23%, 百分比最大回撤为-22.32%, 夏普比率达 1.4。
- 含上一笔赢利过滤版本: 总成交笔数为 1090, 年化收益为 44.34%, 百分比最大回撤为-25.61%, 夏普比率达 1.5。

增加过滤器后, 总成交笔数降低了, 年化收益和夏普比率都提高了, 因此可以推断海龟策略独有的上一笔赢利过滤, 对于日线级别的趋势跟踪策略, 是非常有效的, 其作用在于降低无效交易(即假突破)所造成的亏损, 在整体上提高策略的胜率。该结论在统计学来看, 就是在日线级别中, 连续出现两次大行情的概率极其低, 应对方案就是剔除低概率事件, 从而提高整体胜率。

在日内中高频 CTA 策略中, 该过滤器无效或许是因为 Tick 级别或者分钟级别数据所包含的噪声更多, 故提高连续出现 2 次大行情的概率; 但是日线级别数据所包含的噪声更少, 更能有助于判断趋势, 且趋势本身就非常难以出现连续两次的大行情。

## 6.7 手续费、滑点测试

理论上，基于日线数据的中低频趋势跟踪策略，手续费和滑点是可以忽略不计的。为了验证这个结论，进行对比测试，手续费、滑点测试如图 6-42 所示。

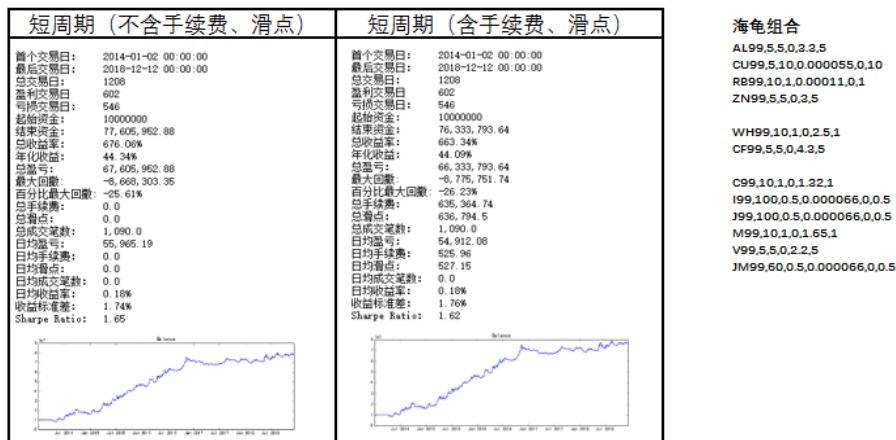


图 6-42 手续费、滑点测试

右图为包含手续费和滑点版本的海龟策略，其中手续费设置为交易所手续费的 1.1 倍，滑点设置为期货合约的最小价格变动。其结果是总手续费为 635 364 元，总滑点为 636 794，结束资金从 77 605 953 元降低至 76 333 794 元，年化收益从 44.34%降低至 44.09%，百分比最大回撤从 -25.61%上升到 -26.23%，夏普比率从 1.65 降低至 1.62，资金曲线形状基本无变化。

综上所述，手续费和滑点对于海龟策略影响不大，在测试中可以忽略不计。

## 6.8 单位头寸限制检验

vn.py 实现了两个维度上的单位头寸限制，分别是：单品种头寸上限是 4，单个方向整体头寸上限是 10。那么现在测试一下适用于国内期货品种的新海龟策略，其最优单位头寸限制数值是否与原版海龟策略一致。

对单品种头寸上限直接分成 3、4、5，然后基于各自的单品种头寸上限对各个方向的整体头寸上限进行细分，分别是 8、9、10、12、14、无限大。故一共构成了  $3 \times 6 = 18$  个测试组合。

在回测中修改单位头寸设置：打开海龟策略文件 turtleStrategy.py，找到第 10 行、第 11 行代码，直接修改其数值即可。

```
MAX_PRODUCT_POS = 4 # 单品种最大持仓
MAX_DIRECTION_POS = 10 # 单方向最大持仓
```

### 1. 分类 1：单品种头寸上限等于 3

基于该分类一共得到 6 个测试组合，其回测效果如图 6-43 所示。

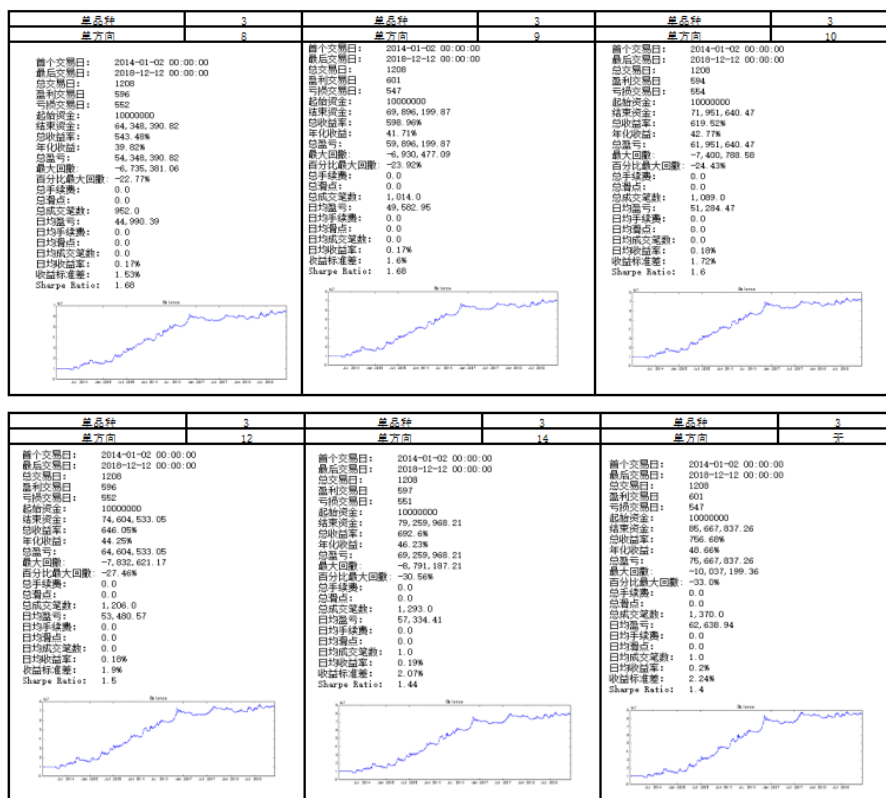


图 6-43 单品种头寸上限为 3 的组合



这 6 个备选组合,按照从左到右,从高到低的顺序,其夏普比率分别是 1.68、1.68、1.6、1.5、1.44、1.4。可以观察到,其数据变化呈现倒“U”形,在单个方向整体头寸上限为 9 时,夏普比率达到顶峰,然后不管是头寸上限增加还是降低,夏普比率都会下降。

另外，单方向头寸从 14 增至无穷大（即限制取消）时，夏普比率降低得并不严重，仅仅为 0.04，这说明针对国内期货品种的海龟策略，其自然达到的单方向头寸单位是大于 14 的，但是其差距并不大。

2. 分类 2: 单品种头寸上限等于 4

对单品种头寸上限为 4 的类型进行回测的效果如图 6-44 所示。

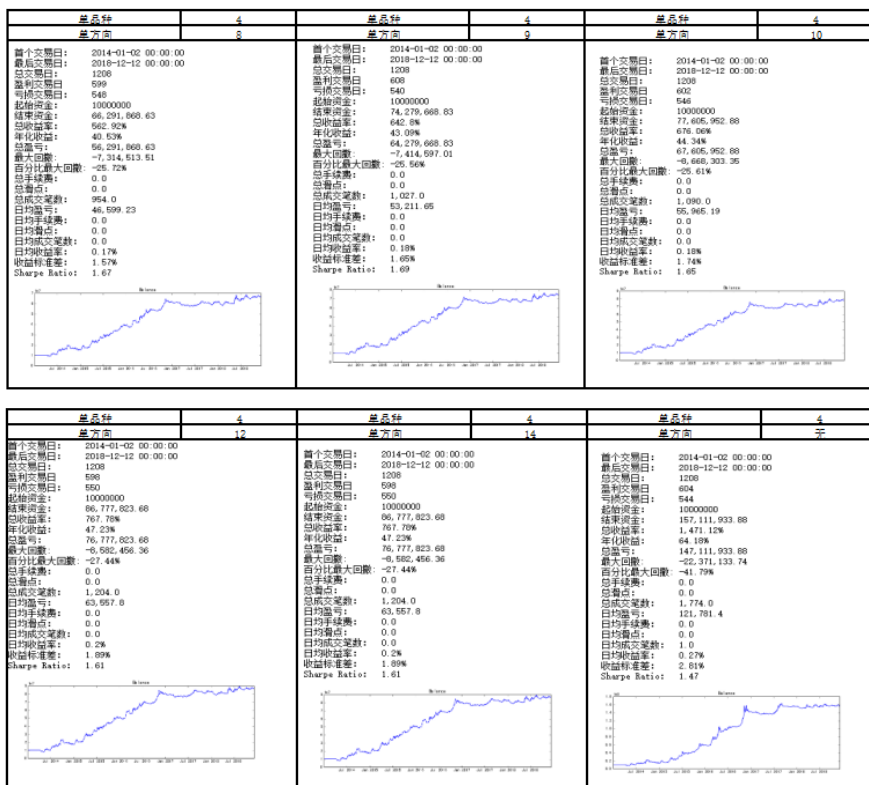


图 6-44 单品种头寸上限为 4 的组合

这 6 个备选组合,按照同样的观察顺序,其夏普比率分别是 1.67、1.69、1.65、1.61、1.62、1.67。其夏普比率变化同样呈现倒“U”形,但是较单位头寸上限为 3 的更加扁平。在单个方向整体头寸上限为 9 时,夏普比率达到顶峰,然后不管是头寸上限增加还是降低,夏普比率都会下降。

另外，从单方向头寸从 14 增大至无穷大（即限制取消）时，夏普比率降低了 0.14。

3. 分类 3: 单品种头寸上限等于 5

对单品种头寸上限为 5 的类型进行回测的效果如图 6-45 所示。

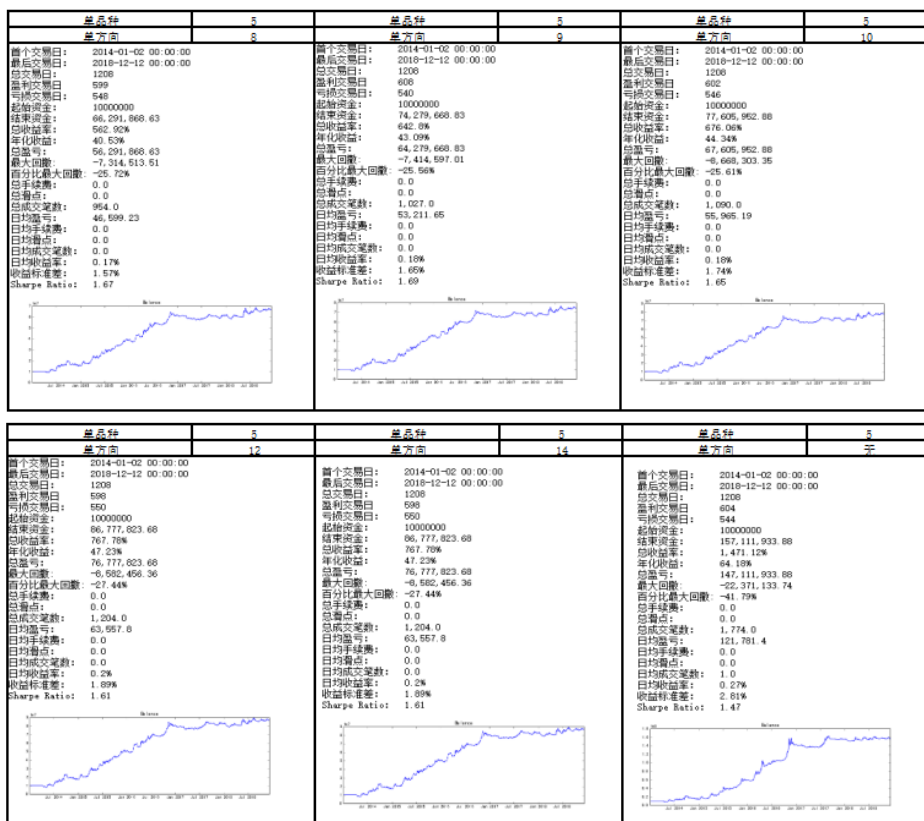


图 6-45 单品种头寸上限为 5 的组合

可以观察到，分类 3 的所有组合与分类 2 的完全一致，这表明单品种头寸单位自然到达的上限是 4，再往上增加其限制已无任何意义。

综上所述，基于海龟组合，单品种头寸单位上限为 4，单个方向整体头寸为 9 时效果最优，其最终版本的海龟策略夏普比率达到 1.69。

## 6.9 关于海龟策略的其他研究方向

目前为止已基本检验完原版海龟策略的关键要素，除了一些细枝末节的逐步加仓检验、两倍 ATR 止损等，最有意思的研究方向是对信号进行优化。

当初的“海龟”们由于时代的局限性只能通过计算机对大量的期货历史行情进行手动计算（即按着计算器，然后用笔记录交易信号的突破位置），所以唐奇安通道因其简单性深受“海龟”们的青睐。

现在的研究方向可以是对信号进行优化，包括增加新的过滤指标，通道方面可以选择布林带通道、金肯特纳通道、多时间周期通道等，然后再加上有效的离场止盈止损指标即可。

信号优化的重点和难点在于参数优化，故需要先写一个优化器，通过排列组合形成一系统参数组合，再调用 CPU 的多线程进行遍历，最后输出每个参数组合对应的策略效果。

# 第 7 章

## 新策略实战

本章先讲述开发新策略、搭建投资组合并进行策略回测的流程，为实战做好准备，还介绍在真实交易情景下接触的三大系统，并且分析造成策略回测与实战中不同结果的原因。

### 7.1 开发新的策略

基于 vn.py 提供的 K 线生成模块、K 线管理模块、CTA 策略模块等，用户可以根据自己的需求来开发新的策略。下面提供一个示例：以 Aroon 指标为主，开发基于 1 分钟沪深 300 股指期货的 CTA 策略。

#### 7.1.1 策略思路

Aroon 指标比较偏门，可以通过阅读相关的券商研报或者国外的金融工程论文来得到比较清晰的策略思路。下面对该指标做一个简单的介绍。

##### 1. Aroon 指标的原理

Aroon 指标是由图莎尔·钱德（Tushar Chande）于 1995 年发明的，它通过计



算自价格达到近期最高值和最低值整个过程中所经过的期间数，帮助投资者预测证券价格趋势强弱及反转等。

Aroon 指标分为两个具体指标，分别为 AroonUp 及 AroonDown。其具体的计算方式为：

- $\text{AroonUp} = [(\text{计算期天数} - \text{最高价后的天数}) / \text{计算期天数}] \times 100$ 。
- $\text{AroonDown} = [(\text{计算期天数} - \text{最低价后的天数}) / \text{计算期天数}] \times 100$ 。
- $\text{Aroon} = \text{AroonUp} - \text{AroonDown}$ 。

## 2. Aroon 指标的用法

当 AroonUp 指标向下跌破 50 时，表示向上的趋势正在失去动力；当 AroonDown 指标向下跌破 50 时，表示向下的趋势正在失去动力；如果两个指标都在低位，则表示股价没有明确的趋势；如果都在 70 以上，则表示趋势十分强烈；如果都在 30 以下，则表示相反的趋势正在酝酿。

若计算期为 20，那么  $\text{AroonUp} = 50$ ，意味着最高价出现在第 10 天， $\text{AroonDown} = 50$ ，意味着最低价出现在第 10 天，则  $\text{AroonUp} - \text{AroonDown} = 0$ ，意味着最高价比最低价较晚出现，离现在近。 $\text{AroonUp} - \text{AroonDown} > x$ ，意味着最高价离现在比最低价离现在多  $20x/100$  天，即若  $x=40$ ， $20x/100=8$  天，表明现在还处于强势中，反之亦然。

一般情况，交易信号的生成标准如下：

- AroonUp 上穿 70，并且  $\text{Aroon} > 0$ ，做多。
- AroonDown 上穿 70，并且  $\text{Aroon} < 0$ ，做空。
- AroonUp 下穿 50，并且  $\text{Aroon} < 0$ ，做空。
- AroonDown 下穿 50，并且  $\text{Aroon} > 0$ ，做多。

从该指标的设计来看，观察某一时点与一段时期内高低位置的距离，来判断这一时点是倾向于涨还是跌，属于趋势类指标。为了更加准确地判断趋势，并且避免一些短期急涨急跌或者小反弹的情形，需要增加新的过滤器。这里会用到



Aroon,  $\text{Aroon} = \text{AroonUp} - \text{AroonDown} > x$ , 意味着拉开了与一段时期内最高价与最低价的距离。

故此过滤器的趋势判断如下所示。

- Aroon 上穿 50, 多头趋势。
- Aroon 下穿-50, 空头趋势。

最后, 结合 Aroon 的交易信号与过滤器, 得到最终的判断标准如下所示。

- AroonUp 上穿 70, 并且 Aroon 上穿 50, 做多。
- AroonDown 上穿 70, 并且 Aroon 下穿-50, 做空。

### 3. 额外的技术指标

CCI 指标用作第二层过滤: 当  $\text{CCI} > 0$  时, 判断为多头趋势; 当  $\text{CCI} < 0$  时, 判断空头趋势。

ATR 指标用于离场: 当 ATR 增大时, 说明市场波动率高, 市场上下跳动的幅度很大, 止损位置可以设置相对远一些; 当 ATR 减少时, 说明行情波动不大, 这时候需要把止损位置设置得近一点。

## 7.1.2 增加 AROON 函数

在 K 线管理模块 ArrayManager 中, 有一部分是调用 TA-Lib 库来生成具体的技术指标的。在此处需要增加 AROON 函数。在 Wing IDE 的 Python Shell 中, 输入下面代码:

```
import talib
help(talib.AROON)
```

这样可以查看 TA-Lib 库中内置的 AROON 函数, 如图 7-1 所示。



```

AROON(...)
    AROON(high, low[, timeperiod=?])

    Aroon (Momentum Indicators)

    Inputs:
        prices: ['high', 'low']
    Parameters:
        timeperiod: 14
    Outputs:
        aroondown
        aroonup

```

图 7-1 TA-Lib 库定义的 AROON 函数

基于 TA-Lib 库内置的 AROON 函数，在 K 线管理模块中构建应用于 K 线时间序列的 AROON 函数，代码如下：

```

def AROON(self, n, array=False):
    '''Aroon指标'''
    aroondown, aroonup = talib.AROON(self.high, self.low, n)

    if array:
        return aroondown, aroonup
    return aroondown[-1], aroonup[-1]

```

Wing IDE 保存退出后，进入“C:\vnpy-1.9.0\vnpy-1.9.0”目录，按下“Shift+鼠标右键”，在此处单击打开命令窗口，输入命令：

```
python setup.py install
```

其目的是更新 Anaconda2 内的 K 线管理模块，以便在外部调用 ArrayManager 时，能正常使用 AROON 函数。（当然，也可以直接去 Anaconda 里面的 vn.py 文件内修改。）

### 7.1.3 策略代码解析

#### 1. 策略参数

aroonWindow, cciWindow, atrWindow 都是通过指定回望窗口大小来生成计算指标的。

```
# 策略参数
aroonWindow = 5          # AROON 窗口数
cciWindow = 16           # CCI 窗口数
atrWindow = 32           # ATR 窗口数
slMultiplier = 5.2       # 计算止损距离的乘数
initDays = 10            # 初始化数据所用的天数
fixedSize = 1            # 每次交易的数量
```

## 2. onBar 函数

- 若当前无仓位，则缓存日高点和日低点。若满足 3 个多头条件，则开仓做多 1 手，反之若满足 3 个空头条件，则开仓做空 1 手。
- 若持有多头仓位，则统计日高点，并且通过 ATR 指标来灵活设置离场点，多头平仓。
- 若持有空头仓位，则统计日低点，并且通过 ATR 指标来灵活设置离场点，空头平仓。

下面是 onBar 函数的具体代码：

```
# 当前无仓位，发送开仓委托
if self.pos == 0:
    self.intraTradeHigh = bar.high
    self.intraTradeLow = bar.low

    # 只有满足 3 个条件才开仓
    if self.cciValue > 0 and self.aroonUp > 70 and self.aroon > 50 :
        self.buy(bar.close + 5, self.fixedSize, True)

    elif self.cciValue < 0 and self.aroonDown > 70 and self.aroon < -50 :
        self.short(bar.close - 5, self.fixedSize, True)

# 持有多头仓位
elif self.pos > 0:
    self.intraTradeHigh = max(self.intraTradeHigh, bar.high)
    self.intraTradeLow = bar.low
    self.longStop = self.intraTradeHigh - self.atrValue * self.slMultiplier

    self.sell(self.longStop, abs(self.pos), True)

# 持有空头仓位
```



```

elif self.pos < 0:
    self.intraTradeHigh = bar.high
    self.intraTradeLow = min(self.intraTradeLow, bar.low)
    self.shortStop = self.intraTradeLow + self.atrValue * self.slMultiplier

    self.cover(self.shortStop, abs(self.pos), True)
...

```

## 7.1.4 策略回测

策略写完后，同样需要更新到 Anaconda2 里的 vn.py 文件里，或者直接复制到“vnpy-1.9.0\examples\CtaBacktesting”文件夹内，方便策略回测。若是希望在模拟盘或者实盘运行，则直接把策略文件放在 VnTrader 文件夹内，然后配置好 CTA\_setting.json 文件即可。

### 1. 回测设置

策略回测是用沪深 300 股指期货指数，其合约规模、手续费、滑点分别是 300、万分之 0.5 和 0.2，回测时间区间是 2010 年至今。

```

# 设置回测使用的数据
engine.setBacktestingMode(engine.BAR_MODE)      # 设置引擎的回测模式为 K 线
engine.setDatabase(MINUTE_DB_NAME, 'IF0000')    # 设置使用的历史数据库
engine.setStartDate('20100416')                 # 设置回测用的数据起始日期
# 配置回测引擎参数
engine.setSlippage(0.2)                          # 设置滑点为股指 1 跳
engine.setRate(0.5/10000)                        # 设置手续费万 0.5
engine.setSize(300)                              # 设置股指合约大小
engine.setPriceTick(0.2)                         # 设置股指最小价格变动
engine.setCapital(1000000)                       # 设置回测本金

```

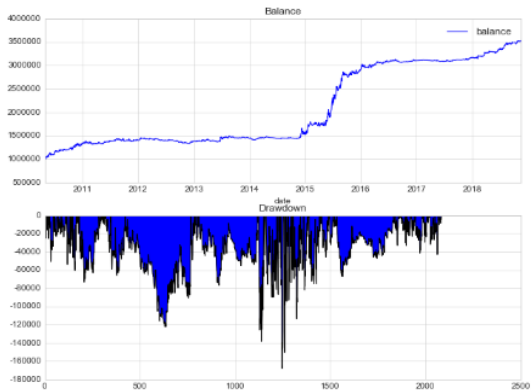
### 2. 回测结果

(1) 逐日统计结果:年化收益为 29.1%，百分比最大回撤-8.35%，夏普比率达 1.26，具体结果如图 7-2 到图 7-4 所示。总体来说，该策略显示出明显趋势跟踪特点，在 2015—2016 年间出现牛熊市，故很好地把握住了大趋势，而其他时间假突破居多，策略表现平庸。

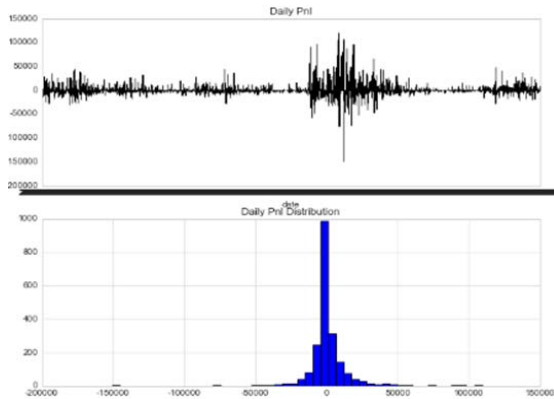


计算按统计结果	
首个交易日:	2010-04-26
最后一个交易日:	2018-11-16
交易日期:	2053
交易日期:	805
亏损交易日:	967
起始资金:	1000000
结束资金:	3,525,920.32
年化收益:	252.59%
日收益率:	29.1%
总收益率:	2,525.92032%
最大回撤:	-167.5321%
百分比最大回撤:	-6.35%
总手续费:	406,219.68
总盈亏:	507,420.0
总资产金额:	8,124,393.660
日均资产:	8,068
日均盈亏:	1,212.64
日均手续费:	195.02
日均盈亏:	243.6
日均总资产:	3,900,333.01
日均总资产:	4.06
日均总资产:	0.068
收益标准差:	0.74%
Sharpe Ratio:	1.26

1. **Introduction**



1. **Introduction**



11. *Journal of the American Medical Association*, 2000; 283: 2689-2696.

(2) 逐笔统计结果胜率为 37.83%，盈亏比达 2.04，逐笔统计显示随着开平次数增大，策略的总体赢利增强，如图 7-5 和图 7-6 所示。

2018-11-17 13:22:33.687000	计算回测结果
2018-11-17 13:22:33.874000	
2018-11-17 13:22:33.875000	第一笔交易: 2010-04-26 11:17:00
2018-11-17 13:22:33.875000	最后一笔交易: 2018-11-16 15:00:00
2018-11-17 13:22:33.875000	总交易次数: 4,229.0
2018-11-17 13:22:33.876000	总盈亏: 2,525,811.41
2018-11-17 13:22:33.876000	最大回撤: -170,198.21
2018-11-17 13:22:33.876000	平均每笔盈利: 597.26
2018-11-17 13:22:33.877000	平均每笔止损: 120.0
2018-11-17 13:22:33.877000	平均每笔佣金: 96.07
2018-11-17 13:22:33.877000	胜率: 37.83%
2018-11-17 13:22:33.877000	盈利交易平均值: 8,123.75
2018-11-17 13:22:33.877000	亏损交易平均值: -3,983.34
2018-11-17 13:22:33.877000	盈亏比: 2.04

图 7-5 逐笔统计结果

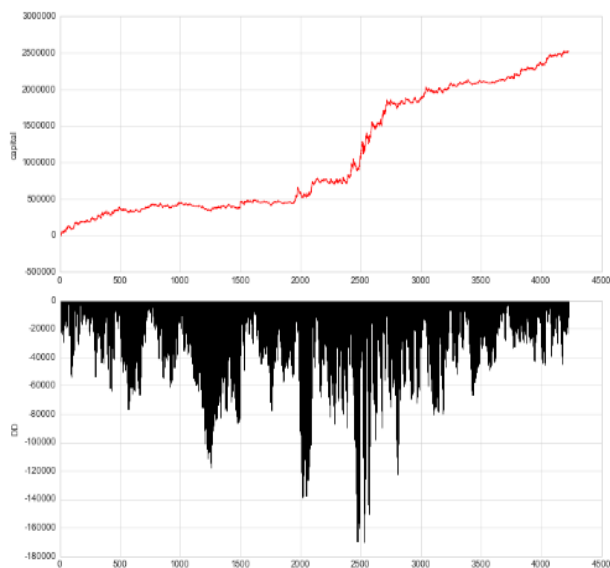


图 7-6 逐笔统计下的资金图和回撤图

## 7.2 多策略的组合回测

根据投资组合理论，通过投资相关性低的品种可以降低非系统性风险，从而提高其收益风险比。在本节中，通过构建由沪深 300 股指期货、螺纹钢期货和橡

股指期货组合的投资组合来进行策略回测。

- 沪深 300 股指期货选用的是滚动回测优化后的 Dual Thrust 策略。
- 螺纹钢期货使用的是滚动回测优化后的布林带通道策略。
- 橡胶期货使用的是滚动回测优化后的多重时间框架策略。

首先回测 2014—2018 年投资组合的历史表现，然后测试优化参数的预测效果。

## 7.2.1 历史表现

### 1. 加载回测函数

创建 BacktestingEngine 对象，用于构建回测函数，设置回测模式、K 线模式、数据库类型、回测/结束时间、合约的滑点、手续费、最小价格变动。然后开始加载策略，运行策略，显示按日统计的结果。函数最终返回的 df 就是用来按日统计结果的。

```
%matplotlib inline

from vnpy.trader.app.ctaStrategy.ctaBacktesting import BacktestingEngine,
MINUTE_DB_NAME

def runBacktesting(strategyClass, settingDict, symbol,
                   startDate, endDate, slippage,
                   rate, size, priceTick):
    """运行单标的回测"""
    engine = BacktestingEngine()                #调用回测引擎
    engine.setBacktestingMode(engine.BAR_MODE)  #回测模式: K 线
    engine.setDatabase(MINUTE_DB_NAME, symbol)  #数据库类型
    engine.setStartDate(startDate)               #回测开始时间
    engine.setEndDate(endDate)                  #回测结束时间
    engine.setSlippage(slippage)                #设置滑点
    engine.setRate(rate)                        #设置手续费率
    engine.setSize(size)                        #设置合约大小
    engine.setPriceTick(priceTick)              #设置最小价格变动

    engine.initStrategy(strategyClass, settingDict) #加载策略
```



```
engine.runBacktesting()           #运行策略
df = engine.calculateDailyResult() #显示按日统计结果
return df                         #返回结果
```

## 2. 沪深 300 股指期货回测设置

沪深 300 股指期货回测设置只交易 1 手，回测时间是 2014—2018 年，滑点是 0.2，手续费是万分之 0.5，合约规模是 300，最小价格变动是 0.2。

```
#沪深 300 股指期货交易 1 手
from strategyDualThrust import DualThrustStrategy
df1 = runBacktesting(DualThrustStrategy, {}, 'IF0000',
                    '20140101', '20180101', 0.2,
                    0.5/10000, 300, 0.2)
```

## 3. 螺纹钢期货回测设置

螺纹钢期货回测设置只交易 10 手，回测时间是 2014—2018 年，滑点是 1，手续费是万分之 10，合约规模是 10，最小价格变动是 1。

```
#螺纹钢期货交易 10 手
from strategyBollChannel import BollChannelStrategy
settingDict = {'fixedSize': 10}
df2 = runBacktesting(BollChannelStrategy, settingDict, 'rb0000',
                    '20140101', '20180101', 1,
                    0.1/1000, 10, 1)
```

## 4. 橡胶期货回测设置

橡胶期货回测设置只交易 5 手，回测时间是 2014—2018 年，滑点是 5，手续费是万分之 0.45，合约规模是 10，最小价格变动是 5。

```
#橡胶期货交易 5 手
from strategyMultiTimeframe import MultiTimeframeStrategy
settingDict = {'fixedSize': 5}
df3 = runBacktesting(MultiTimeframeStrategy, settingDict, 'ru0000',
                    '20140101', '20180101', 5,
                    0.45/10000, 10, 5)
```

## 5. 显示组合回测

沪深 300 股指期货、螺纹钢期货和橡胶期货的测量结果以 DataFrame 的形式



缓存起来，故其投资组合就是 3 个 DataFrame 相加即可，即  $dfp = df1 + df2 + df3$ ，然后去掉空值 `dropna()`。

然后创建回测引擎，设置回测投资组合策略要用到的起始资金，用 `calculateDailyStatistics()` 计算逐日统计的结果，然后通过 `showDailyResult()` 显示回测结果。

```
# 合并获得组合回测结果
dfp = df1+ df2 + df3

# 注意如果被抛弃的交易日位于回测的前后，即两者不重合的日期中，则不会影响组合曲线的正确性
# 但是如果被抛弃的交易日位于回测的中部，即两者重合的日期中，则组合曲线会出现错误（丢失交易日）
dfp = dfp.dropna()

# 创建回测引擎，并设置组合回测初始资金后，显示结果
engine = BacktestingEngine()
engine.setCapital(1000000)          #本金为 100 万元
dfp, result = engine.calculateDailyStatistics(dfp)
engine.showDailyResult(dfp, result)
```

逐日统计结果显示多策略组合，其年化收益为 22.46%，百分比最大回撤 -2.48%，夏普比率达 2.53，如图 7-7 所示。由于交易品种相关性不高，有利于抵消了各自的非系统性风险，投资组合的夏普比率高于单个品种的夏普比率。

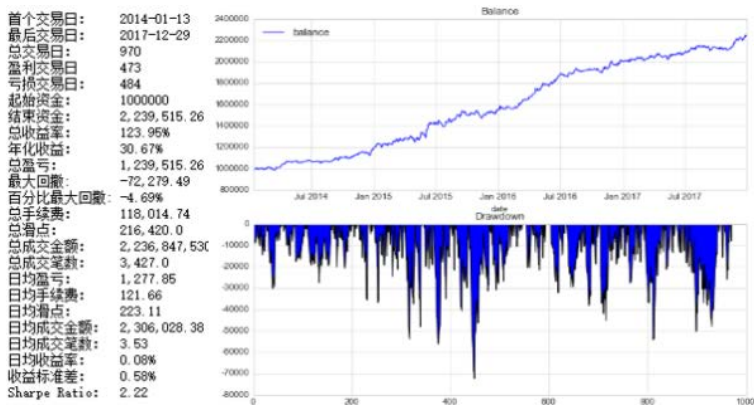


图 7-7 多策略组合回测结构



## 7.2.2 预测表现

2018 年以后的预测表现：年化收益为 30.73%，百分比最大回撤-3.76%，夏普比率达 2.41，如图 7-8 所示。预测效果表现出很高的夏普比率，故可上模拟盘测试，先用 SimNow 账号连接 CTP 运行策略，若一段时间模拟测试表现良好，就可以换上期货公司的实盘账号动用小资金去运行策略了。

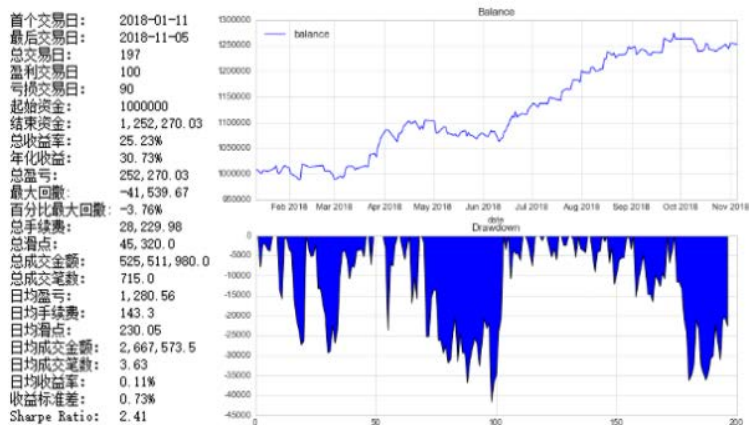


图 7-8 2018 年以后的预测表现

## 7.2.3 回测的注意事项

为了让策略回测接近实盘，必须注意手续费问题。手续费分为期货公司手续费和交易所手续费

### 1. 期货公司手续费

期货公司手续费相当于股票中的佣金。对股票来说，炒股的费用包括印花税、佣金、过户费及其他费用。相对来说，从事期货交易的费用就只有手续费。期货交易手续费是指期货交易者买卖期货成交后按成交合约总价值的一定比例所支付的费用。

目前国内有四大期货交易所，共 22 个上市品种，不同品种手续费不一样。各

期货公司都是交易所的会员，客户参与期货交易的手续费有固定的一部分上交给交易所，另一部分由期货公司收取，期货公司收取的标准是在期货交易所的基础上加一部分，用于自身的运营。不同地区的不同期货公司收取的手续费是不一样的，相对大的、实力强的期货公司手续费高一些，而一些小的期货公司手续费略低。手续费也会因客户资金量的大小、交易量的多少而不同，对于资金量大的甚至成百上千万元的客户，期货公司也会适度降低手续费。

但总的来说，期货公司手续费相对交易所收取的手续费少很多，故在策略回测中不考虑其手续费的影响。

## 2. 交易所手续费

下面展示四大期货交易所关于其交易所品种及其所收取的手续费。

(1) 上海期货交易所（简称上交所）品种，如表 7-1 所示。

表 7-1 上交所品种

交易所	品种	代码	交易所手续费	日内单/双	平今手续费
能源中心	原油	SC	20元	单边	
上海期货交易 所	螺纹钢	rb	万1	双边	
	热轧卷板	hc	万1	双边	
	镍	ni	6元	单边	1/5/9合约双边
	铜	cu	万0.5	单边	
	锌	zn	3元	单边	
	铝	al	3元	单边	
	锡	sn	1元	单边	1/5/9合约3元
	铅	pb	万0.4	单边	
	黄金	au	10元	单边	
	白银	ag	万0.5	双边	
	线材	wr	万0.4	双边	
	橡胶	ru	万0.45	双边	
	沥青	bu	万1	双边	
	燃料油	fu	万0.5	单边	

(2) 郑州商品交易所 (简称郑商所) 品种, 如表 7-2 所示。

表 7-2 郑商所品种

交易所	品种	代码	交易所手续费	日内单/双	平今手续费
郑州商品交易所	玻璃	FG	3元	双边	6元
	动力煤	ZC	4元	双边	
	甲醇	MA	2元	双边	
	锰铁	SM	3元	双边	6元
	硅铁	SF	3元	双边	9元
	菜油	OI	2元	单边	
	白糖	SR	3元	单边	
	白糖期权	SR	1.5元	双边	
	PTA	TA	3元	单边	
	强麦	WH	2.5元	单边	
	棉纱	CY	4元	单边	
	棉花	CF	4.3元	单边	
	菜粕	RM	1.5元	单边	
	苹果	AP	20元	双边	
	粳稻	JR	3元	双边	
	晚籼稻	LR	3元	双边	
	普麦	PM	5元	双边	
	早籼稻	RI	2.5元	双边	
	油菜籽	RS	2元	双边	

(3) 大连商品交易所 (简称大商所) 品种, 如表 7-3 所示。

表 7-3 大商所品种

交易所	品种	代码	交易所手续费	日内单/双	平今手续费
大连商品交易所	铁矿石	i	万0.6	双边	1/5/9合约 万0.6
	焦炭	j	万0.6	双边	1/5/9合约 万1.8
	焦煤	jm	万0.6	双边	1/5/9合约 万1.8
	胶板	bb	万1	单边	
	纤板	fb	万1	单边	
	聚乙烯	v	2元	单边	
	玉米	c	0.2元	单边	1/5/9合约 1.2元
	淀粉	cs	1.5元	双边	0.75元
	豆一	a	2元	双边	
	豆二	b	2元	双边	
	塑料	l	2元	双边	1元
	豆粕	m	0.2元	双边	1/5/9合约 0.75元, 其 他0.1元
	豆粕期权	m	1元	双边	0.5元
	棕榈油	p	2.5元	双边	1.25元
	豆油	y	2.5元	双边	1.25元
	PP	pp	万0.6	双边	万0.3
	鸡蛋	jd	万1.5	双边	

(4) 中国金融交易所(简称中金所)品种,如表 7-4 所示。

表 7-4 中金所品种

交易所	品种	代码	交易所手续费	日内单/双	平今手续费
中国金融交易所	沪深300	IF	万0.23	双边	万6.9
	中证500	IC	万0.23	双边	万6.9
	上证50	IH	万0.23	双边	万6.9
	10年国债	T	3元	单边	
	5年国债	TF	3元	单边	
	2年国债	TS	3元	单边	

从表 7-1 到表 7-4 中可以发现平仓手续费比开仓手续费高,其原因是为了防止过度投机。其中,中金所甚至启动平今惩罚机制,其开仓手续费为万分之 0.23,但是平仓手续费高达万分之 6.9。为了规避这种惩罚机制,需要用锁仓来代替平今。下面来对比这两种方式的特点。

正常日内开仓-平仓手续费率如下。

- 多头开仓手续费率为 0.0023%。
- 日内平多的手续率为 0.069%。

故当日开平仓一次,损失的手续费率为  $0.0023\% + 0.069\% = 0.00713\%$ 。可以通过锁仓规避高昂的平今仓手续费,即把日内平多变成空头开仓,并且在第二天开盘后平掉昨天的多仓与空仓:

- 多头开仓手续费为 0.0023%。
- 空头开仓手续费为 0.0023%。
- 多头平昨手续费为 0.0023%。
- 空头平昨手续费为 0.0023%。

当日平仓改为锁仓模式,手续费降低至 0.0092%。由于 vn.py 可以用底层算法进行锁仓,大大增加了策略开发的简便性。

在回测中,锁仓模式交易笔数是平仓手数的 2 倍,但考虑到加入锁仓模式规避手续费并且交易手数不变,可以把股指期货手续费调整到 0.005%。



最后应该注意中金所在计算平今交易手续费时，对当日平仓成交按成交时间先后顺序，逐笔采取“先平当日新开仓，再平历史仓”的方式计算平今数量，每笔成交的平今数量为该笔成交中平当日新开仓数量。故在无今仓情况下，应该先平掉昨仓，再开今仓。

### 3. 底层委托转换

在 `vn.py` 的 `vtEngine` 中实现本地持仓数据缓存功能：基于持仓查询、成交推送、委托挂撤单来实时计算当前所有仓位的持仓量、冻结量和可用量的数据。通过缓存对象实现委托开平仓自动转换功能，将上层的开平仓请求根据实时持仓情况转化成底层需要发出的委托，主要提供以下三种模式。

#### （1）普通模式（`MODE_NORMAL`）：

- 什么转换都不做，直接发出。
- 大部分合约默认使用该模式。

#### （2）上期所模式（`MODE_SHFE`）：

- 将平仓委托自动根据今仓与昨仓情况拆分，优先平今仓，今仓不足则平昨仓。
- 上期所合约默认使用该模式。

#### （3）平今惩罚模式（`MODE_TDPENALTY`）：

- 反向有昨仓且无今仓时优先平昨仓。
- 反向有今仓或无昨仓时改为开仓。
- 禁止平今仓。
- 需要在“`vnpy-1.9.0\examples\VnTrader`”里的 `VT_setting.json` 里配置使用该模式的合约，其配置如图 7-9 所示。

```

1  {
2      "fontFamily": "微软雅黑",
3      "fontSize": 10,
4
5      "mongoHost": "localhost",
6      "mongoPort": 27017,
7      "mongoLogging": true,
8
9      "darkStyle": true,
10     "language": "chinese",
11
12     "logActive": true,
13     "logLevel": "debug",
14     "logConsole": true,
15     "logFile": true,
16
17     "tdPenalty": ["IF", "IH", "IC"],
18
19     "maxDecimal": 4
20 }

```

图 7-9 设置平今惩罚模式

下面展示底层委托转换函数。

```

def convertOrderReq(self, req):
    """转换委托请求"""
    # 普通模式无须转换
    if self.mode is self.MODE_NORMAL:
        return [req]

    # 上期所模式拆分今昨，优先平今
    elif self.mode is self.MODE_SHFE:
        # 开仓无须转换
        if req.offset is OFFSET_OPEN:
            return [req]

        # 多头
        if req.direction is DIRECTION_LONG:
            posAvailable = self.shortPos - self.shortPosFrozen
            tdAvailable = self.shortTd - self.shortTdFrozen
            ydAvailable = self.shortYd - self.shortYdFrozen
        # 空头
        else:
            posAvailable = self.longPos - self.longPosFrozen
            tdAvailable = self.longTd - self.longTdFrozen
            ydAvailable = self.longYd - self.longYdFrozen

    # 平仓量超过总可用，拒绝，返回空列表

```

```

    if req.volume > posAvailable:
        return []
    # 平仓量小于今可用, 全部平今仓
    elif req.volume <= tdAvailable:
        req.offset = OFFSET_CLOSETODAY
        return [req]
    # 平仓量大于今可用, 先平今仓再平昨仓
    else:
        l = []

        if tdAvailable > 0:
            reqTd = copy(req)
            reqTd.offset = OFFSET_CLOSETODAY
            reqTd.volume = tdAvailable
            l.append(reqTd)

        reqYd = copy(req)
        reqYd.offset = OFFSET_CLOSEYESTERDAY
        reqYd.volume = req.volume - tdAvailable
        l.append(reqYd)

    return l

# 平今惩罚模式, 没有今仓则平昨仓, 否则锁仓
elif self.mode is self.MODE_TDPENALTY:
    # 多头
    if req.direction is DIRECTION_LONG:
        td = self.shortTd
        ydAvailable = self.shortYd - self.shortYdFrozen
    # 空头
    else:
        td = self.longTd
        ydAvailable = self.longYd - self.longYdFrozen

    # 这里针对开仓和平仓委托使用同一套逻辑

    # 如果有今仓, 则只能开仓 (或锁仓)
    if td:
        req.offset = OFFSET_OPEN
        return [req]
    # 如果平仓量小于昨可用量, 全部平昨仓
    elif req.volume <= ydAvailable:

```



```
        if self.exchange is EXCHANGE_SHFE:
            req.offset = OFFSET_CLOSEYESTERDAY
        else:
            req.offset = OFFSET_CLOSE
        return [req]
    # 平仓量大于昨可用量, 平仓再反向开仓
    else:
        l = []

        if ydAvailable > 0:
            reqClose = copy(req)
            if self.exchange is EXCHANGE_SHFE:
                reqClose.offset = OFFSET_CLOSEYESTERDAY
            else:
                reqClose.offset = OFFSET_CLOSE
            reqClose.volume = ydAvailable

            l.append(reqClose)

        reqOpen = copy(req)
        reqOpen.offset = OFFSET_OPEN
        reqOpen.volume = req.volume - ydAvailable
        l.append(reqOpen)

    return l

# 其他情况则直接返回空
return []
```

## 7.3 模拟测试

### 7.3.1 策略文件目录

VnTrader 的 CTA 策略模块, 支持加载当前运行时目录下的策略文件。注意策略文件必须以小写的 `strategy` 开头 (如 `strategyMultiTimeframeRu.py`), 同时文件中的策略类名称中必须包含大写的 `Strategy` (如 `MultiTimeframeRuStrategy`)。



现在将投资组合的 3 个策略文件放在“vnpy-1.9.0\examples\VnTrader”文件夹内，策略类分别命名为 MultiTimeframeRuStrategy，BollChannelRbStrategy 和 DTIfStrategy，然后配置好 CTASetting.json 文件即可。

### 7.3.2 实盘/模拟盘配置文件

对于同一个策略类，可以通过设定不同的参数和交易品种来创建不同的实例，用户可以在“vnpy-1.9.0\examples\VnTrader”文件夹内的 CTA\_setting.json 中配置实例。

每个实例的配置项如下所述。

- name: 实例名字，不能重复。
- className: 策略类名，基于具体的策略类来创建新的“子实例”。命名规则可以是“策略类\_品种”+“Strategy”，如 BollChannel\_RbStrategy。
- vtSymbol: 实例交易的品种。
- 策略中定义的参数，在这里没有给出的参数将使用策略里赋予的默认值。

因此，配置内容如图 7-10 所示。

```
1  [
2      {
3          "name": "DaulThrustNew",
4          "className": "DT_IfStrategy",
5          "vtSymbol": "IF1812"
6      },
7
8      {
9          "name": "BollChannel",
10         "className": "BollChannel_RbStrategy",
11         "vtSymbol": "rb1901",
12         "fixedSize": 10
13     },
14
15     {
16         "name": "MultTimeFrame",
17         "className": "MultiTimeframe_RuStrategy",
18         "vtSymbol": "ru1901",
19         "fixedSize": 5
20     }
21 ]
```

图 7-10 实盘/模拟盘配置

打开 VnTrader，连接交易接口后，单击菜单栏的“功能→CTA 策略”可以打开 CTA 策略模块的实盘/模拟盘交易管理界面，如图 7-11 所示。

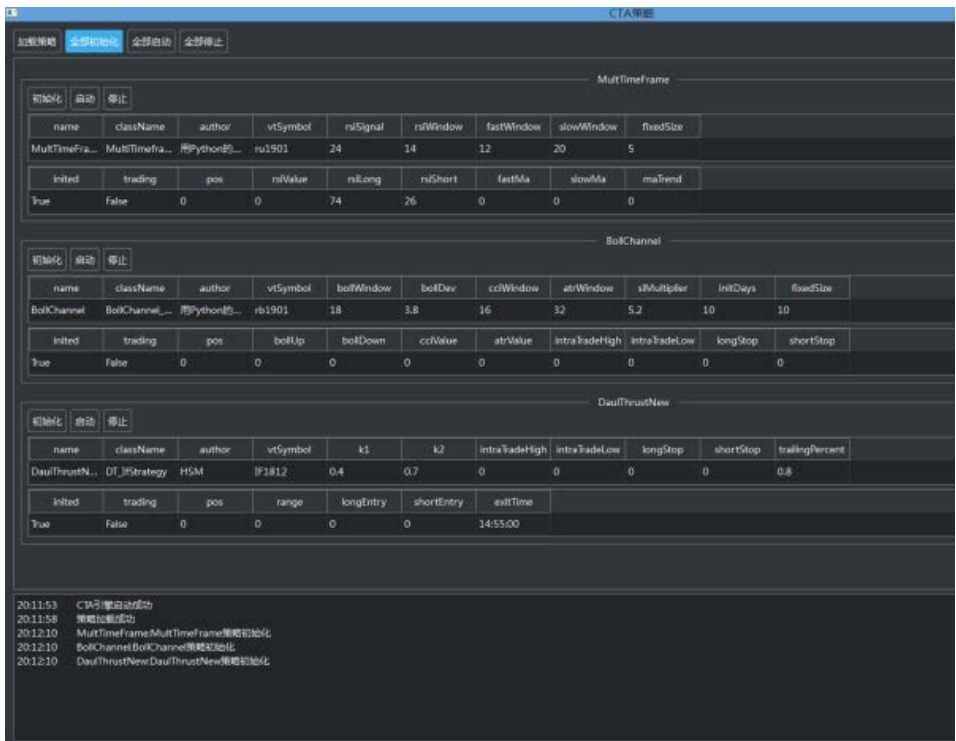


图 7-11 实盘/模拟盘交易管理界面

其中各区域作用如下：

- 单击“加载策略”按钮，策略引擎会读取 CTA\_setting.json 中的策略配置，生成具体的策略实例（下面称为“策略”），并显示在管理界面上。
- 单击“初始化”按钮会调用策略的 onInit 函数，并完成策略开始交易前的初始化操作，此时策略可以计算参数指标等，但是无法发出委托。
- 单击“启动”按钮会调用策略的 onStart 函数，正式启动策略的交易功能，此时策略基于收到的行情、成交、委托等数据进行相应的逻辑判断，并发出委托。

- 单击“停止”按钮会调用策略的 onStop 函数，撤销当前策略已发出的所有未成交委托，并停止策略的交易功能，恢复到无法发出委托的状态。
- 如果加载了多个策略，窗口最上方的“全部初始化”“全部启动”和“全部停止”可以一次性对所有的策略执行相应的操作，省去多次点击。
- 策略参数栏用于显示策略参数、状态，以及策略中指定输出的中间变量值，以监控策略运行。
- 最下方的 Log 输出窗口，策略可调用 writeCtaLog() 函数在这里输出。

另外，对于同一个策略类，可以通过设定不同的参数和交易品种来创建不同的实例，以跨时间周期策略 MultiTimeframeStrategy 为例，下面定义了 MultiTimeframeStrategy 策略的两个实例：一个用于交易螺纹钢 Rb，另一个用于交易橡胶 Ru，它们的名称 name 与具体策略参数都不同。如图 7-12 所示。

```

1  [
2      {
3          "name": "DaulThrustNew",
4          "className": "DT_IfStrategy",
5          "vtSymbol": "IF1812"
6      },
7
8      {
9          "name": "MultiTimeFrame_Rb",
10         "className": "MultiTimeframeStrategy",
11         "vtSymbol": "rb1901",
12         "fixedSize": 10,
13         "rsiWindow": 20,
14         "rsiLength": 14,
15         "fastWindow": 8,
16         "slowWindow": 10
17     },
18 },
19
20 {
21     "name": "MultiTimeFrame_Ru",
22     "className": "MultiTimeframeStrategy",
23     "vtSymbol": "ru1901",
24     "fixedSize": 5,
25     "rsiWindow": 24,
26     "rsiLength": 14,
27     "fastWindow": 12,
28     "slowWindow": 20
29 }
30 ]

```

图 7-12 多策略实例实现

实现效果如图 7-13 所示。

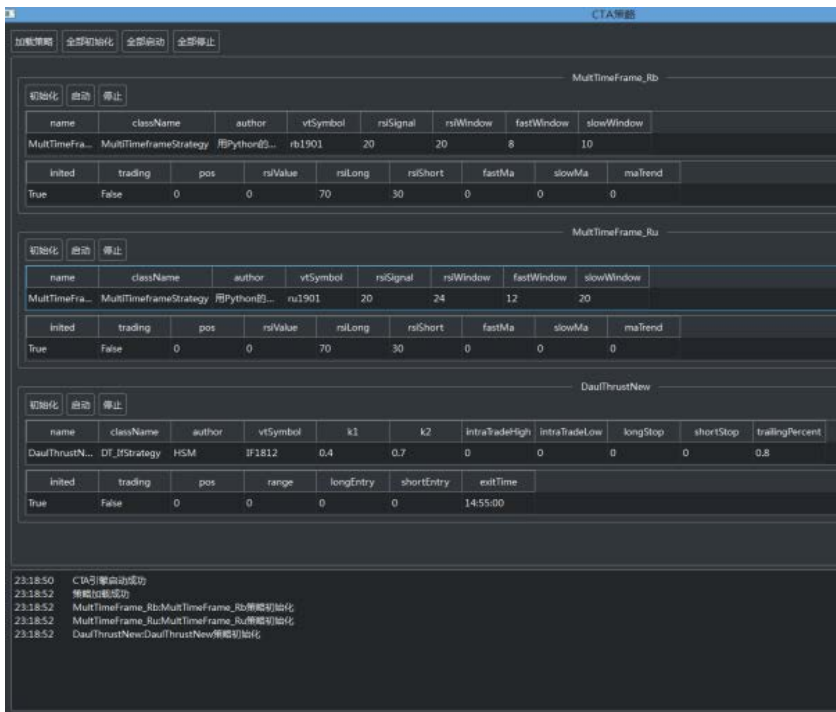


图 7-13 监控组件的多策略实例

## 7.4 真实交易环境

### 7.4.1 交易环境的 3 套系统

国内市场存在 3 套系统：交易所系统、经纪商柜台系统和投资者交易系统。这 3 套系统启动是有先后顺序的，一定是交易所系统先启动完毕后，再启动经纪商柜台系统，最后才启动投资者的交易系统。

- **交易所系统**：先启动核心交易，比如撮合队列的报盘前置机；然后启动行情服务器，用于不断获取交易所撮合队列的实时队列并推送到交易所系统行情客户端。

- **经纪商柜台系统**：经纪商主要指期货公司和证券公司，而经纪商柜台系统指的就是 CTP、飞马、飞创等柜台系统。其主要功能包括先启动柜台服务器，用于期货公司接受客户订单，进行验资验券，前端风控等；然后启动经纪商行情服务器，用于交易所系统行情客户端，把行情推送到经纪商行情服务器上。关于行情服务器，不同技术公司有不同优势，例如郑商所易盛公司开发的易盛柜台对郑商所品种有速度的优势；飞马柜台则对股指期货品种进行专门的优化。
- **投资者交易系统**：先初始化交易系统，连接柜台和行情服务器，确认上一个交易日的结算报告，查询合约、资金、持仓数据，订阅合约行情。然后初始化策略应用模块，例如在 CTA 策略交易模块上，分别单击加载策略和初始化策略启动策略。正常情况是，在下午收盘后 15~20 分钟最后一个结算 Tick 推送完毕，就可以停止策略。若不停止策略，在非交易时段就会从柜台系统那边收到一堆奇怪的行情数据，这是因为经纪商有可能测试系统或者升级系统，从而推送异常数据。

## 7.4.2 交易环境的数据流

交易所系统发出 Tick 行情，中间经过经纪商柜台系统转发到投资者交易系统上的这一过程根据硬件设备的不同，传递时间也不相同。若投资者的机器在托管机房，则这个时间大概有几十微秒到几百微秒；若从公网接入，那么时间可以从几毫秒增长到几十毫秒，这是最慢，也是最常见的方案，即从网卡接收数据，途径内核到用户态内存。

从用户态内存拿到数据，在投资者交易系统内做计算，然后发出订单，订单指令到网卡，通过网络到达经纪商柜台系统，经纪商柜台系统会先进行验资验券：不符合条件则直接拒单；符合条件把委托发送到交易所系统，插入中央订单簿等待撮合成交。之后，交易所系统的回报信息也会通过经纪商柜台系统推送到投资者交易系统里，如图 7-14 所示。

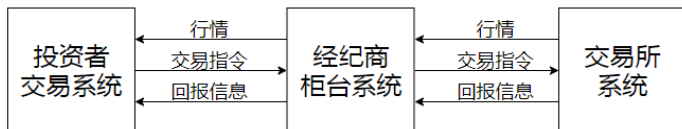


图 7-14 交易环境的数据流

## 7.5 实际操作注意事项

在模拟盘中，用户可能发现现实交易的情况远没有回测时出色，下面归纳出 7 点原因。（通过 `vn.py` 回测已杜绝未来函数出现的可能性，因为从数据库里载入数据后，会逐条推送到策略中，以模拟真实交易环境。）

### 7.5.1 计算错误

在回测中需要设置合约规模、最小价格变动、滑点、手续费等。若 1 个甚至多个参数设置错误，则会导致回测出现巨大的误差。比如回测沪深 300 股指期货行情数据时却用到螺纹钢合约的数据，尽管在回测中或许能得到非常喜人的夏普比率，但是在仿真/实盘交易中肯定亏钱。

正确设置如下：

```
engine.setBacktestingMode(engine.BAR_MODE) # 设置引擎的回测模式为 K 线
engine.setDatabase(MINUTE_DB_NAME, 'IF0000') # 设置沪深 300 股指期货 1 分钟 K 线数据
engine.setStartDate('20100416') # 设置回测用的数据起始日期

engine.setSlippage(0.2) # 设置滑点为股指 1 跳
engine.setRate(0.5/10000) # 设置手续费万 0.5
engine.setSize(300) # 设置股指合约大小，IF 股指是 300 元/点
engine.setPriceTick(0.2) # 设置股指最小价格变动，沪深 300 股指最新价格变化单位是 0.2
engine.setCapital(1000000) # 设置回测本金为 100 万元
```

错误设置如下：

```
engine.setBacktestingMode(engine.BAR_MODE) # 设置引擎的回测模式为 K 线
engine.setDatabase(MINUTE_DB_NAME, 'IF0000') # 设置沪深 300 股指期货 1 分钟 K 线数据
```



```

engine.setStartDate('20100416')          # 设置回测用的数据起始日期

engine.setSlippage(1)                    # 设置滑点为股指 1 跳
engine.setRate(1/1000)                  # 设置手续费万 10
engine.setSize(10)                      # 设置股指合约大小
engine.setPriceTick(1)                  # 设置股指最小价格变动
engine.setCapital(1000000)              # 设置回测本金为 100 万元

```

## 7.5.2 数据使用误差

在期货领域，策略回测用的是期货指数生成信号，交易的是期货指数，但是在实际中用主力连续合约生成信号，交易的也是主力连续合约。两者有一定的区别，故实际效果会比策略回测时要差一些。

### 1. 主力连续合约

主力连续合约由期货品种中持仓量最大的合约的 K 线连接而成，它最能代表当前行情的走势，但随着交割期的接近，大量的投机资金会退出并转移到另一个合约，形成新的主力合约。图 7-15 展示的是天勤终端把每一时期的主力合约的走势拼成一幅连续图，所以这就造成换月时的价格跳空。

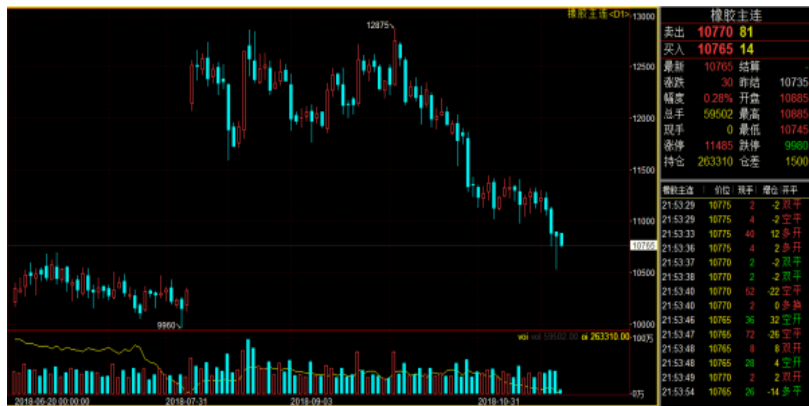


图 7-15 主力连续合约跳空



## 2. 期货指数

期货指数是指根据期货品种中的每个合约（以持仓量或者成交量作为加权指数）加权计算出的平均价格。随着持仓量不断往远月转移，指数权重也相应地后移。如图 7-16 所示，相对于主力连续合约来说，期货指数就会有很好的客观性和连贯性，避免了换月带来的跳空缺口，这个特点也意味着期货指数更能全面反映实际价格变动情况。



图 7-16 期货指数连续合约

综上所述，主力连续合约因为有换月的状况所以有跳空情况，而期货指数合约是全部合约的加权，所以会有很优秀的连续性。主力连续合约的优势自然是可作为短期内精准的分析工具，劣势是换月时出现跳空；期货指数的优势恰是主力连续合约的劣势，即连续性较好。降低数据使用误差的方法是策略加载到指数上触发信号，而下单的话直接指定主力连续，因为只交易主力合约。

另外一点值得注意的是，商品期货分为日盘和夜盘。若策略是仅仅基于日盘数据得到的，那么将该策略用在夜盘时间的话表现也会大打折扣。

## 7.5.3 过拟合

从图 7-17 可以看出，一个荒谬的模型，只要足够复杂，也可以完美适应数据，这就是过拟合。过拟合的成因有两方面：一是样本内数据特征与样本外数据特征



差异较大；二是模型太复杂，而样本数量太少。

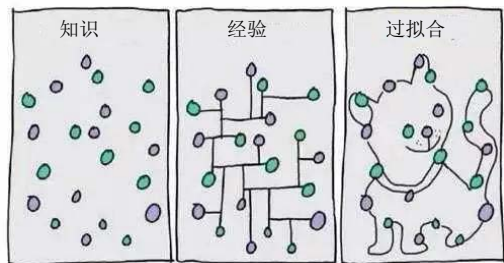


图 7-17 过拟合

在量化交易领域，回测是基于历史数据的，但历史数据的样本是有限的，如果交易策略的参数过多，或者交易逻辑过于复杂，导致策略模型过多地适应历史数据。量化策略的建模本质就是一个从大量的貌似随机的数据中找寻局部非随机数据的过程，如果不借助统计学的知识，则很容易落入过拟合的陷阱。

解决过拟合也应该从成因着手，即收集多样化的样本，不要使用太复杂的模型和交叉检验。交叉检验的基本思想是把在某种意义下将原始数据进行分组，一部分作为训练集，另一部分作为验证集，首先用训练集对分类器进行训练，再利用验证集来测试训练得到的模型，以此来作为评价分类器的性能指标。

## 7.5.4 幸存者偏差

幸存者偏差指的是，当取得资讯的渠道仅来自“幸存者”时（因为无法由“死者”获得来源），此资讯可能会存在与实际情况不同的偏差。

幸存者偏差的情况常见于投资理财类的节目或文章，例如，因为投资理财类电视节目仅邀请投资成功者上节目谈论其成功投资的经验，观众会将该成功投资者投资的方式，视为高成功率的投资方式，但观众并不会在电视节目中看到以相同或类似投资方式但最后失败的投资者，因而高估此投资方式的成功概率。实际上人们接收的信息其实是经过筛选后的结果，因此，幸存者偏差造成的结果就是往往一开始就忽略了大量的数据或样本，导致基于幸存者偏差的结论偏离实际。

在策略回测中，幸存者偏差表现为回测结果好，很有可能就是运气成分罢了，而不是真正把握住了市场的本质。如图 7-18 所示，在上百次随机交易中，总会有表现非常好的资金曲线，这些策略有可能是抓住了交易的“圣杯”，但更大的可能是运气好。

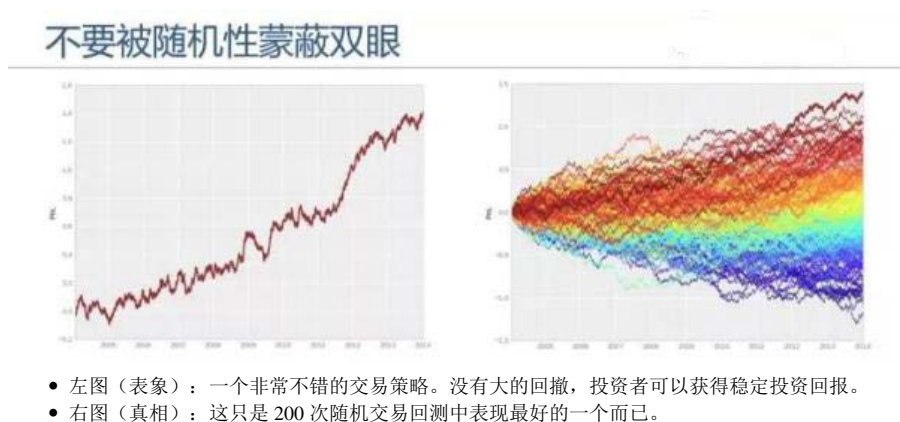


图 7-18 随机交易回测效果

金融市场并不缺乏明星，特别在牛市中会涌现出一大批明星交易员、明星基金经理，但是真正存活下来，能够稳定赢利的却少之又少，因此需警惕幸存者偏差，回测策略的成功很有可能就是运气好而已。

### 7.5.5 策略周期

从投资的角度，每种策略都有其适应的场景。在合适的场景下选到了适合的资产，那么策略会表现非常出色。但实际情况金融市场是轮动的，资产配置随大的金融周期轮动，股票市场随着行业板块轮动。若在回测的时候选对了风口，赶上了趋势，而实盘时候错过风口或者选错了金融资产，就会事与愿违。

比如，在 2016 年、2017 年表现出色的价值投资策略，在 2018 年的表现却让人大跌眼镜，净值屡创新低。就算是非常平庸的 CTA 策略，在 2015 年牛熊市带来的大行情也能赚到盆满钵满，对于单边趋势行情，所有 CTA 策略都赚钱；但是

对于震荡行情，所有 CTA 策略都不赚钱。

交易策略的局限性在于其只关注比较微观的信息，需要加上人为从更宏观的视角来判断局势，比如哪些策略需要启动，哪些策略需要停止；在策略持续亏损时，也需要判断是市场环境发生变化导致的，还是策略本身出现问题等。

## 7.5.6 动态变化的现实环境

在策略回测的过程中，冲击成本、滑点、手续费和自身对市场的影响是很难被精确模拟的。因为在现实交易环境中，这些因素会不断变化。

- **冲击成本**：冲击成本应该从供求关系方面着手分析，大笔地买会推高价格，大笔地卖会使价格降低，这使得我们实际的交易成本和下单的价格是存在差异的。要减少这个差异，一方面可以减少每次下单的数量，将大单拆分成小单；另一方面在设计回测系统的时候模拟撮合部分要写复杂些。
- **滑点**：滑点主要发生在流动性不好的品种身上，例如策略模型发出交易信号，但是实际上得不到理想的价格，甚至被迫用对手价来成交。还有可能是市场上高频机构的交易速度提升了一个档次，导致的结果是，本来在买 3 价肯定能成交的，要延后到买 6 价才能百分之百成交。故在真实交易环境中，滑点是非常难以估算的。
- **手续费**：手续费问题一般发生于政策性调整，目的在于降低投机热潮，因此是不可预知的，在回测中也非常难以模拟。2015 年调整的股指期货的 100 倍日内平均手续费，几乎逼走所有高频做市商和高频套利的人，他们都转战到“黑色系”商品期货，只有部分日内波动和中低频策略投资者生存下来。2017 年 7 月“黑色系”商品期货开始被猛炒，焦炭、焦煤的平今仓手续费上调至 3 倍，铁矿石平今仓手续费上调到 2 倍。
- **自身对市场的影响**：在实盘做交易的时候，因为市场有众多的参与者。整个市场处于一个动态博弈的状态，每个投资者的每一项操作，都可能会引发一连串的连锁反应，如同蝴蝶效应一般是完全没有办法模拟的。



现实交易环境是复杂的，也是很难在回测环境中模拟的，所以只有深刻了解金融市场、了解市场运作的原理，才能规避真实交易环境与回测环境中的差异点。

### 7.5.7 人为干预

理想的策略执行情况表现在把交易策略应用到实盘时，宽客应该充分地相信模型，并且严格地执行。每当遇到回撤的时候，依然坚持用模型来执行策略。

人为干预的因素会给策略执行带来不可控的结果，让策略效果变好或者变坏全依赖于干预者的宏观判断和长期经验。特别是当面对策略运行带来的持续回撤时，是坚信模型的策略，还是立刻放弃原策略、启动新策略，抑或对策略参数进行微调，很多时候只能见仁见智。

故在真实环境中赚钱的关键，技术是一方面，自信与坚持则是另一方面。



# 附录A

## 主流交易品种

本章主要介绍目前在金融领域各主流交易品种的定义、特点、交易所、撮合规则，以及适用的交易策略。

### A.1 股票

#### A.1.1 股票的定义

股票是一种有价证券，是股份有限公司在筹集资本时向出资人公开发行的、用以证明出资人的股本身份和权利，并根据股票持有人所持有的股份数享有权益和承担义务的可转证的书面凭证。股票代表其持有人（即股东）对股份公司的所有权，每一股股票所代表的公司所有权是相等的，即“同股同权”。

简单地说，股票对应的是对公司的所有权，只要公司不倒闭，股票还是有价值的。但是即使公司临近倒闭，在理论上股票价值也并不是接近零的。因为公司的股东（债务人）和债权人会进行博弈。在这种博弈中，公司股东具有绝对的强势地位，而债权人则被动承受所有风险，故公司股东会采取激进做法让公司有机会起死回生，此时股票表现出风险有限而利润无限的特点，股票价值也会新增实



物期权的性质。

在现实中，有倒闭风险的公司也容易成为多空双方资金博弈的场所，股价变化呈现极大的不可预测性，如 2018 年年初复牌后的乐视网股份，频繁地出现涨停及跌停。

## A.1.2 股票交易所

### 1. 上海证券交易所

上海证券交易所（以下简称“上交所”，英文缩写为 SSE）1990 年成立于上海市浦东新区。主要交易股票集中在主板，上市股票 1479 只，同时又以央企、国企为主，股票代码一般为以 600 开头的六位代码。600 开头的股票是上证 A 股，属于大盘股，其中 6006 开头的股票是最早上市的股票，6016 开头的股票为大盘蓝筹股。

上交所市场交易时间为每周一至周五。上午为前市，9:15 至 9:25 为开盘集合竞价时间，9:30 至 11:30 为连续竞价时间。下午为后市，13:00 至 15:00 为连续竞价时间。所以一天的交易时长为 4 小时。

### 2. 深圳证券交易所

深圳证券交易所（以下简称“深交所”，英文缩写为 SZSE）1990 年成立于广东省深圳福田区。上市股票 2115 只，分别登陆于主板（股票代码以 000 开头）、中小企板（股票代码以 002 开头）、创业板（股票代码以 300 开头）。其中也有一些小众的，如股票代码以 200 开头的股票是深证 B 股；股票代码以 400 开头的股票是三板市场股票。

深交所市场交易时间为每周一至周五。上午为前市，9:15 至 9:25 为开盘集合竞价时间，9:30 至 11:30 为连续竞价时间。下午为后市，13:00 至 14:57 为连续竞价时间，14:57 至 15:00 为收盘集合竞价时间。



### A.1.3 股票竞价规则

在上海证券交易所和深圳证券交易所中，产生股票价格的方式有两种，一是开盘集合竞价，二是开盘后的连续竞价。但是其撮合成交规则必须符合成交时价格优先、时间优先的原则。

成交时价格优先的原则为较高价格买入申报优先于较低价格买入申报，较低价格卖出申报优先于较高价格卖出申报。

成交时时间优先的原则为买卖方向、价格相同的，先申报者优先于后申报者。先后顺序按交易主机接受申报的时间确定。

#### 1. 集合竞价

集合竞价是将数笔委托报价或一时段内的全部委托报价集中在一起，根据不高于申买价和不低于申卖价的原则产生一个成交价格，且在这个价格下成交的股票数量最大，并将这个价格作为全部成交委托的交易价格。集合竞价的基本过程：设股票 G 在开盘前分别有 5 笔买入委托和 6 笔卖出委托，根据价格优先的原则，按买入价格由高至低和卖出价格由低至高的排列顺序将如图 A-1 所示。

买单序号	买入价格(元)	数量	-	卖单序号	卖出价格(元)	数量
1	3.80	2	-	1	3.52	5
2	3.76	6	-	2	3.57	1
3	3.65	4	-	3	3.60	2
4	3.60	7	-	4	3.65	6
5	3.54	6	-	5	3.70	6
				6	3.75	3

图 A-1 5 笔买入委托和 6 笔卖出委托

按不高于申买价和不低于申卖价的原则，首先可成交第一笔，即 3.80 元买入委托和 3.52 元的卖出委托，若要同时符合申买者和申卖者的意愿，则其成交价格必须在 3.52 元与 3.80 元之间，但具体价格要视以后的成交情况而定。第一笔成交



后的委托情况如图 A-2 所示。

买单序号	买入价格(元)	数量	-	卖单序号	卖出价格(元)	数量
1	3.80	2	-	1	3.52	5
2	3.76	6	-	2	3.57	1
3	3.65	4	-	3	3.60	2
4	3.60	7	-	4	3.65	6
5	3.54	6	-	5	3.70	6
				6	3.75	3

图 A-2 第一笔成交后的委托情况

在第一次成交中，由于卖出委托的数量多于买入委托，按交易规则，序号 1 的买入委托 2 手全部成交，序号 1 的卖出委托还剩余 3 手。第二笔成交情况：序号 2 的买入委托价格为不高于 3.76 元，数量为 6 手。在卖出委托中，序号 1~3 的委托的数量正好为 6 手，其价格意愿也符合要求，正好成交，其成交价格 在 3.60 元~3.76 元的范围内，成交数量为 6 手。应注意的是，第二笔成交价格的范围是在第一笔成交价格的范围之内，且区间要小一些。 第二笔成交后的委托情况如图 A-3 所示。

买单序号	买入价格(元)	数量	-	卖单序号	卖出价格(元)	数量
3	3.65	4	-			
4	3.60	7	-	4	3.65	6
5	3.54	6	-	5	3.70	6
			-	6	3.75	3

图 A-3 第二笔成交后的委托情况

第三笔成交情况：序号 3 的买入委托价格要求不超过 3.65 元，而卖出委托序号 4 的委托价格符合要求，这样序号 3 的买入委托与序号 4 的卖出委托就正好配对成交，价格为 3.65 元，因卖出委托数量大于买入委托，故序号 4 的卖出委托仅



只成交了 4 手。第三笔成交后的委托情况如图 A-4 所示。

买单序号	买入价格(元)	数量	-	卖单序号	卖出价格(元)	数量
4	3.60	7	-	4	3.65	2
5	3.54	6	-	5	3.70	6
			-	6	3.75	3

图 A-4 第三笔成交后的委托情况

完成以上三笔委托后，因最高买入价为 3.60 元，而最低卖出价为 3.65，买入价与卖出价之间再没有相交部分，所以这一次的集合竞价就已完成，最后一笔的成交价就为集合竞价的平均价格。剩下的其他委托将自动进入开盘后的连续竞价。

在以上过程中，通过一次次配对，成交的价格范围逐渐缩小，而成交的数量逐渐增大，直到最后确定一个具体的成交价格，并使成交量达到最大。在最后一笔配对中，如果买入价和卖出价不相等，其成交价就取两者的平均。在这次的集合竞价中，三笔委托共成交了 12 手，成交价为 3.65 元，按照规定，所有成交的委托无论是买入还是卖出，其成交价都定为 3.65 元，交易所发布的股票 G 的开盘价就为 3.65 元，成交量 12 手。

当股票的申买价低而申卖价高最终导致没有股票成交时，上交所就将连续竞价后产生的第一笔价格作为开盘价。而深交所对此却另有规定：若最高申买价高于前一交易日的收盘价，就选取该价格为开盘价；若最低申卖价低于前一交易日的收盘价，就选取该价格为开盘价；若最低申买价不高于前一交易日的收盘价、最高申卖价不低于前一交易日的收盘价，则选取前一交易日的收盘价为今日的开盘价。

## 2. 连续竞价

连续竞价的成交方式与集合竞价有很大的区别，它是在买入的最高价与卖出的最低价的委托中一一对地成交，其成交价为申买与申卖的平均价。现仍以股票 G 为例，某一时刻委托报价的排序情况如图 A-5 所示。

买单序号	买入价格(元)	数量	-	卖单序号	卖出价格(元)	数量
1	3.80	2	-	1	3.52	5
2	3.76	6	-	2	3.57	1
3	3.65	4	-	3	3.60	2
4	3.60	7	-	4	3.65	6
5	3.54	6	-	5	3.70	6
			-	6	3.75	3

图 A-5 5 笔买入委托和 6 笔卖出委托

委托买入的最高价为序号 1 的 3.80 元，卖出最低价为序号 1 的 3.52 元，这一对优先成交，其价格为两者的平均值， $(3.80 + 3.52)/2$ ，故产生的价格为 3.66 元，成交数量只有 2 手。交易所发布的即时行情为成交价 3.66 元、数量 2 手。这一笔成交后的委托情况如图 5-6 所示。

买单序号	买入价格(元)	数量	-	卖单序号	卖出价格(元)	数量
			-	1	3.52	3
2	3.76	6	-	2	3.57	1
3	3.65	4	-	3	3.60	2
4	3.60	7	-	4	3.65	6
5	3.54	6	-	5	3.70	6
			-	6	3.75	3

图 A-6 第一笔成交后的委托情况

第二笔，序号 1 的卖出价为 3.52 元，序号 2 的买入价为 3.76 元，这一对可以成交，成交价格为两者的平均值，价格为 3.64 元，数量为 3 手。第二笔成交后的委托情况如图 A-7 所示。

买单序号	买入价格(元)	数量	-	卖单序号	卖出价格(元)	数量
2	3.76	3	-	2	3.57	1
3	3.65	4	-	3	3.60	2
4	3.60	7	-	4	3.65	6
5	3.54	6	-	5	3.70	6
			-	6	3.75	3

图 A-7 第二笔成交后的委托情况

第三笔，序号 2 的卖出委托与序号 2 的买入委托可以成交，成交均价为 3.67 元，成交量 1 手。第三笔成交后的委托情况如图 A-8 所示。



买单序号	买入价格(格)	数量	-	卖单序号	卖出价格(元)	数量
2	3.76	2	-			
3	3.65	4	-	3	3.60	2
4	3.60	7	-	4	3.65	6
5	3.54	6	-	5	3.70	6
			-	6	3.75	3

图 A-8 第三笔成交后的委托情况

第四笔，序号 3 的卖出价为 3.60 元，序号 2 的买入价为 3.76 元，这一对可以成交，成交价格为两者的平均值，价格为 3.68 元，数量为 2 手。第四笔成交后的委托情况如图 A-9 所示。

买单序号	买入价格(元)	数量	-	卖单序号	卖出价格(元)	数量
3	3.65	4	-			
4	3.60	7	-	4	3.65	6
5	3.54	6	-	5	3.70	6
			-	6	3.75	3

图 A-9 第四笔成交后的委托情况

第五笔，序号 4 的卖出价为 3.65 元，序号 3 的买入价为 3.65 元，这一对可以直接成交，成交价格为 3.65 元，数量为 4 手。到目前为止无法撮合成交了，第五笔成交后的委托情况如图 A-10 所示。

买单序号	买入价格(元)	数量	-	卖单序号	卖出价格(元)	数量
4	3.60	7	-	4	3.65	2
5	3.54	6	-	5	3.70	6
			-	6	3.75	3

图 A-10 第五笔成交后的委托情况

### A.1.4 T+1 制度

我国实行的“T+1”制度本质上是证券交易交收方式，使用的对象有 A 股、基金、债券、回购交易，是指达成交易后，相应的证券交割与资金交收在成交日的下一个营业日（T+1 日）完成。

世界上各大主要股票市场普遍实行 T+0 制度。但是这种随时买卖的交易方式其实存在比较大的限制。以美国市场为例，美股账户与国内账户一样，也是分为现金账户和融资账户的，美股现金账户实际上是 T+3 交易；如果是融资账户，资金在 2.5 万美元以下的是 T+1 交易，但是每 5 个交易日有 3 次 T+0 的机会，超过 3 次，账户就会被锁定 90 天不能进行互联网买卖交易；如果账户资金超过 2.5 万美元，则可以无限制地进行 T+0 交易。但是美国证券交易佣金的收取方式和中国的不同，是按交易笔数而不是交易股票数量收取的，一般一笔是 5 美元左右。

T+1 制度产生的根源为了保护证券市场，让证券市场更健康，能为更多公司融资。下面阐述 T+1 制度的优劣势。

#### T+1 的优势

- (1) 抑制投机，无法快进快出。
- (2) 主力虚假交易有所减少。
- (3) 在单次交易成本确定的情况下，减少了散户的交易次数，在一定程度上减少了散户的交易成本；
- (4) 减少散户亏损金额，频繁交易是证券投资大忌，在对股市没有较深的理解的情况下频繁交易大多亏损，据北美证券监管者协会研究结果统计，超过 70% 的人都会在“T+0”中亏损，仅有约 10% 的回转交易者才能从中获利。此外，回转交易者必须实现 56% 的年收益率，才能弥补手续费及保证金利息，更不用说资本利得税。

#### T+1 的劣势

- (1) 对散户影响比对主力影响大：由于主力具备资金优势，因此主力可以做变相的“T+0”，而散户操作变相的“T+0”难度较大。
- (2) 不能及时出逃：如果当天不小心在高位买入，那么在出现股价一路下跌时，当天无法及时出逃。这一点对个人和机构都是一样的，很多机构当天买入“爆



雷”的股票也一样无可奈何。

(3) 增强了机构的控盘能力：当小盘股较多筹码集中在主力手中时，由于剩下的在散户手中的筹码每天最多交易一次，无疑降低了主力控盘的难度。涨跌停板的限制也降低了控盘难度，给了各庄家资金缓冲时间。

对比来看，无论是 T+0 还是 T+1，对谁都是一样的，但庄家都能依靠其专业性和资金量形成一定优势。T+0 更适合成熟市场，就中国现在以散户为主的市场而言，T+1 明显更合适。

## A.1.5 股票交易策略

### 1. 股票市场中性策略

股票市场中性策略又称 Alpha 策略，是当前国内私募证券投资基金最常用的策略之一。它从消除市场系统性风险（Beta）的维度出发，通过同时构建多头和空头头寸对冲市场风险，以期获得较稳定的绝对收益。国内通常的操作方式为在买入股票的同时卖空与股票等市值的股指期货（也可以采取融券方式）。

虽然量化策略的最终目标是通过各种手段获取稳定的 Alpha，但是由于当今市场政策，完全对冲 Beta 成本过大，而且期权标的物大都是大盘股，不能代表整个市场。以上种种原因导致产品无法进行完全对冲，所以现实中多数产品的收益是由 Alpha 与 Beta 两部分组成的。Beta 提供的收益来自产品对市场及风格（大小盘、成长价值等）的敞口，Alpha 提供的收益来自产品管理人的管理能力，是产品收益剔除了 Beta 带来的收益后所剩余的部分。所以很多打着“Alpha 策略”标志的产品最终的业绩表现实际上是由 Alpha 与 Beta 共同决定的。通过量化的手段，产品仅能够获取稳定的 Alpha，但是 Beta 带来的收益并不稳定。只有在产品 Beta 敞口与市场偏好相符时，Beta 才会进一步增强产品的收益。反之，Beta 敞口会明显削弱产品的收益。考虑到 Beta 对于收益的影响强于 Alpha，市场整体表现的变化或者市场风格的切换都可能对于产品最终的业绩表现产生明显的影响。

在实际中经常使用的 Alpha 策略主要有多因子、风格轮动、行业轮动、资金流、动量反转等。其中以多因子最为常见，该策略选择一系列因子搭建模型，通过这些因子筛选股票，满足则买入，不满足则卖出。多因子的最大优势在于，在不同的市场和行情下，因子库中总有一些因子能够发挥作用。

## 2. 日内回转策略

日内回转策略，又称日内 T+0，是指投资者就同一个标的（如股票）在同一个交易日内各完成一次买进和卖出的行为。简单地说，就是当日买进的股票在当日卖出，或者当日卖出的股票再在当日买进的交易行为。因此，日内回转交易可以概括为“买入现券—融券卖出一现券还券”和“融券卖出一买入还券—偿还融券”，在日内反复捕捉其波动率，而其所投资的股票也绝对不会在持仓中“过夜”。这种交易策略是在国内融资融券交易机制相对成熟后才逐渐兴起的，而这类群体主要由曾经在国内熬夜炒美股的日内交易团队转化而来。T+0 团队做日内回转策略，需要跟机构投资者或者券商大散户合作来提供底仓，做 T+0 不会改变底仓股票的数量和品种。

日内回转策略根据持仓时间可以分成三种：日内超短、日内趋势、日内长线。

- **日内超短**：持仓时间大多在两分钟内，一般不会超过 10 分钟，赢利的仓位偶尔会长点，也很少超过 30 分钟的，这种策略的止损价格设置在千分之三。
- **日内趋势**：持仓时间一般在 30 分钟左右，因为想要拿的波段大点，所以止损价位一般也比日内超短的止损价格设置得大一些，如千分之六。
- **日内长线**：持仓时间一般在一个小时以上，赚取的利润很大、风险也大，回撤不易控制，对技术盘口要求最高，止损价格一般设置在百分之一左右。

由于做 T+0 可以降低股票持仓成本，而且与日间中长线策略有较低的相关性，所以可以与市场中性策略配合使用，用于增加 Alpha 策略的年化收益。



## A.2 期货

### A.2.1 期货的定义

期货是在交易所内进行交易的标准化合约，该合约约定了在未来某一个特定的时间以确定的价格交易一定数量的某种商品。期货具有以下特点。

(1) 合约标准化，即除价格外，期货合约的所有条款都是由期货交易所预先规定好的，交易双方不需对交易的具体条款进行协商，给期货交易带来极大便利，节约交易时间，减少交易纠纷。

(2) 交易集中化，即期货交易必须在期货交易所内进行。

(3) 双向交易和对冲机制。双向交易，既可以买入开仓，又可以卖出开仓。与双向交易的特点相联系的还有对冲机制，即买入开仓后可以卖出平仓，卖出开仓后可以买入平仓。期货交易的双向交易和对冲机制这一特点，吸引了大量期货投资者参与交易，因为在期货市场上，投资者有双重的获利机会。

(4) 杠杆机制。期货交易实行保证金制度，也就是说交易者在进行期货交易时需缴纳少量的保证金，一般为合约交易价值的 5%~10%，就能完成数倍乃至数十倍的合约交易。

(5) 每日无负债结算制度，也叫“逐日盯市”，就是在每个交易日结束后，对交易者当天的盈亏状况进行结算，在不同交易者之间根据盈亏进行资金划转。如果交易者亏损严重，保证金账户资金不足时，则要求交易者必须在下一个交易日开市前追加保证金，以做到“每日无负债”。

### A.2.2 期货交易所

#### 1. 中国金融期货交易所

中国金融期货交易所（以下简称“中金所”，英文缩写为 CFFEX）于 2006



年 9 月在上海正式挂牌成立。交易品种只有股指期货和国债期货,股指期货在 2015 年推出时的交易量一度成为四大期货交易所之首。但在股指受限后成交量急速萎缩,黑色金属系成为期货市场新宠,但由于其金融期货的特殊性,成为股票对冲风险必备的工具。

中金所合约为代码(大写)+年份+月份,注意,合约代码只有中金所和郑商所是大写的,其合约如下。

- 沪深 300 股指期货: IF1808。
- 中证 500 股指期货: IC1808。
- 上证 50 股指期货: IH1808。
- 5 年期国债期货: TF1809。
- 10 年期国债期货: T1809。

股指期货的交易时间为 9:30-11:30(前市)和 13:00-15:00(后市),国债期货的交易时间为 9:15-11:30(前市)和 13:00-15:15(后市),如图 A-11 所示。

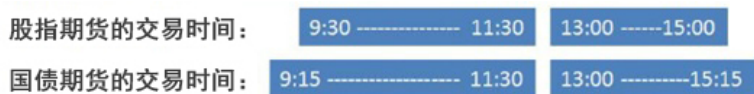


图 A-11 中金所交易时间

## 2. 上海期货交易所

上海期货交易所(以下简称“上期所”,英文缩写为 SHFE)于 1999 年 9 月在上海正式运行。主要交易品种为金属期货,同时也涉及能源期货和化工期货。合约品种代码(小写字母)+年份+月份,部分合约如下。

- 金属期货:铜(cu1808),铝(al1808),锌(zn1808),铅(pb1808),镍(ni1808),锡(sn1908),黄金(au1808)白银, (ag1808),螺纹钢(rb1808),线材(wr1808),热轧卷板(hc1808)。
- 能源期货:燃料油(sc1808),沥青(bu1808)。



- 化工期货：天然橡胶（ru1808）。

这里要注意，上期所品种不仅有日盘，也有夜盘，如图 A-12 所示。

商品	21:00-23:00	橡胶、螺纹钢、热轧卷板、沥青
期货	21:00 -----23:30	大豆、豆粕、豆油、菜籽粕、菜籽油、棕榈油、棉花、白糖、焦煤、焦炭、铁矿石、动力煤、玻璃、甲醇、PTA
夜盘	21:00 ----- 01:00	铜、铝、锌、铅、镍、锡
	21:00 ----- 02:30	黄金、白银
交易时间	法定节假日（不包含双休日）前第一个工作日没有夜盘	

图 A-12 上期所夜盘时间

### 3. 大连商品交易所

大连商品交易所（以下简称“大商所”，英文缩写为 DCE）于 1993 年大连正式运行。其主要交易品种为农产品期货，同时也涉及工业品期货。合约品种代码（小写字母）+年份+月份，部分合约如下。

- 农产品期货：玉米（c1809）玉米淀粉（cs1809），黄大豆 1 号（a1809），黄大豆 2 号（b1809），豆粕（m1809），豆油（y1809），棕榈油（p1808），纤维板（fb1808），胶合板（bb1808），鸡蛋（jb1808）。
- 工业品期货：聚乙烯（l1808），聚氯乙烯（v1808），聚丙烯（pp1808），焦炭（j1808），焦煤（jm1808），铁矿石（i1808）。

### 4. 郑州商品交易所

郑州商品交易所（以下简称“郑商所”，英文缩写为 ZCE）成立于 1990 年 10 月 12 日，是经国务院批准成立的国内首家期货市场试点单位，在现货交易成功运行两年以后，于 1993 年 5 月 28 日正式推出期货交易。与大商所差不多，其主要交易品种也是农产品期货，同时涉及工业品期货。合约品种代码（大写字母）+年份+月份，部分合约如下。

- 农产品期货: 强麦 (WH809), 普麦 (PM809), 一号棉花 (CF809), 白糖 (SR809), 菜油 (OI809), 早籼稻 (RI809), 油菜籽 (RS809), 菜籽粕 (RM809), 粳稻 (JR809), 晚籼稻 (LR809), 棉纱 (CY809), 苹果 (AP809)。
- 工业品期货: PTA (TA809), 甲醇 (MA809), 玻璃 (FG809), 硅铁 (SF809), 锰硅 (SM808), 动力煤 (ZC809)。

## A.2.3 期货交易策略

### 1. CTA 策略

商品交易顾问 (Commodity Trading Advisor, 简称 CTA) 是指通过为客户提供期货、期权方面的交易建议, 或者通过受管理的期货账户参与实际交易, 来获得收益的机构或个人。传统意义上, CTA 基金的投资品种仅限于商品期货, 但近年已扩展到包括利率期货、股指期货、外汇期货在内的几乎所有期货品种。CTA 基金因为很好的业绩稳定性, 以及与其他策略的低相关性, 赢得了快速发展。总体而言, CTA 基本上能够分三大类, 其中包括, 趋势跟踪约占 70%, 均值回归占 25% 左右, 逆势或趋势反转占 5% 左右。

#### (1) 趋势跟踪策略

趋势跟踪策略是基于市场并非有效的假设, 基本面变化的信息需要一定的传递时间, 资产价格不能立即反映基本面的变化, 价格向合理方向逐渐变化的过程所表现出的趋势。与正态分布相比, 资产收益率的分布通常具有“尖峰肥尾”的特点, “肥尾”提供了趋势跟踪策略的收益。趋势跟踪策略的赢利与市场的波动性密切相关, 在一定程度就是“追涨杀跌”的策略, 通过快速止损实现“亏小赢大”的局面, 从而在整体上获利。

#### (2) 价差套利策略

价差套利策略通过捕捉市场的不合理价差, 买入被低估的资产, 卖出被高估的资产, 获得回归收益, 达到资本赢利或避险的目的。价差套利交易风险小, 回



报稳定，对于大资金而言，如果单边重仓介入，将面临持仓成本较高、风险较大的不足，反之，如果单边轻仓介入，虽然可能降低风险，但其机会成本、时间成本也较高。因此整体而言，大资金单边重仓抑或单边轻仓介入期市，均难以获得较为稳定和理想的回报。而大资金如以多空双向持仓介入期市，也就是进行套利交易，则既可回避单边持仓所面临的风险，又可能获取较为稳定的回报。进一步细分，价差套利大致又可以分为跨期套利、期现套利、跨品种套利、跨市场套利这四种。

### （3）反趋势策略

相对趋势跟踪策略追踪趋势，反趋势策略预测拐点，通常运用头肩形态、突破形态、交易量等反转指标来发现趋势的转折信号，然后建立头寸。但是在实际运用中，反趋势策略非常小众。

## 2. 高频交易策略

高频交易策略是指从那些人们无法利用的、极为短暂的市场变化中寻求获利的自动化程序交易，比如某种证券买入价和卖出价差价的微小变化，或者某只股票在不同交易所之间的微小价差。这种交易的速度如此之快，以至于有些交易机构将自己的“服务器群组”安置到了离交易所的服务器很近的地方，以缩短交易指令通过光缆传送的时间。一般是以高频做市商/套利算法进行非常高速的证券交易，从中赚取证券买卖价格的差价。

在国内，因为受到政策和交易所的限制，无法做到欧美高频交易那样“闪电下单”，所以国内高频策略能做的只有以下3类。

### （1）高频做市商

这种策略在交易所挂限价单进行双边交易，以提供流动性。所谓双边交易，是指做市商手中持有一定存货，然后同时进行买和卖两方交易。这种策略的收入包括买卖价差、交易所提供的返佣和固定佣金。

### （2）无风险套利

套利策略注重两种高相关性的产品之间的价差。比如，一个股指 ETF 的价格，理论上应该等于组成该 ETF 的股票价格的加权平均。但因种种原因，有时我们会发现市场上这两种价格并不一致，此时即产生套利机会，可以买入价低一方，同时卖出价高一方，以赚取差价。随着市场流动性的增强，这种机会发生的次数和规模会越来越小，并且机会往往转瞬即逝，因此需要借助高频交易的技术来加大搜寻的规模以抓住交易时机。

### （3）短趋势策略

短趋势策略预测一定时间内的价格走势。相对于低频的趋势策略，高频交易的主要数据源是比 Tick 级别数据更精确的交易订单簿（Order Book Events），所以可以在委托单的粒度上进行分析 and 预测来抓大单的动向。Tick 级别数据其实就是一种对交易订单簿的降采样，其前提假设是：最佳买卖价是最重要的信息，以丢弃其他相对不重要的信息为代价，缩减数据规模，让数据处理变得更容易。

## A.3 期权

### A.3.1 期权的定义

期权是指一种能在未来某特定时间以特定价格买入或卖出一定数量的某种特定商品的权利。它是在期货的基础上产生的一种金融工具，给予买方购买或出售标的资产的权利。期权的持有者在该项期权规定的时间内有选择买或不买、卖或不卖的权利，可以实施该权利，也可以放弃该权利，而期权的出卖者则只有期权合约规定的义务。简单来说，相对于期货买卖双方都拥有的权利和义务，在期权这里，买方拥有权利（选择权），而卖方是必须承担义务。

期权又分为看涨期权（Call）和看跌期权（Put），根据买卖方向又可以分为多头（Long）和空头（Short）。所以就产生了以下 4 种基本的交易形式。



## 1. 买入看涨期权 (Long Call)

支付权利金，一旦标的物价格在未来一段时间内上涨，可以履行看涨期权，利润是到期日价格 ( $S$ ) 与执行价 ( $K$ ) 的差额，即  $S-K$ ，如图 A-13 所示。当标的物价格在一段时间内下跌，则期权买入者不能行使其选择权。买入看涨期权在预期价格上涨的前提下表现为有限的风险和无限的利润。

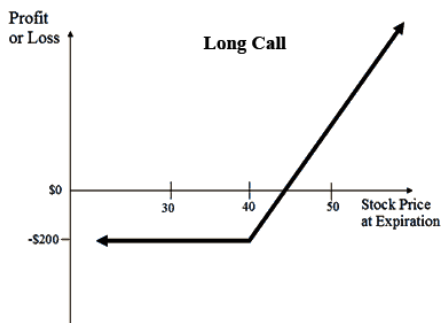


图 A-13 买入看涨期权

## 2. 卖出看涨期权 (Short Call)

获得权利金，一旦标的物价格在未来一段时间内上涨，期权买入者可以履行看涨期权，此时其损失为到期日价格 ( $S$ ) 与执行价 ( $K$ ) 的差额，即  $-(S-K)$ ，或者  $K-S$ ，如图 A-14 所示。当标的物价格在一段时间内下跌，期权买入者不能行使其选择权，赚的就是权利金。卖出看涨期权在预期价格下跌的前提下表现为有限的利润和无限的风险。

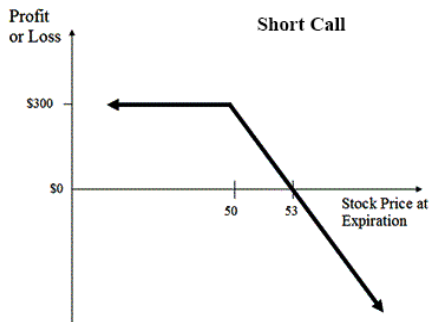


图 A-14 卖出看涨期权

### 3. 买入看跌期权 (Long Put)

支付权利金，一旦标的物价格在未来一段时间内下跌，可以履行看跌期权，利润是到期日价格 ( $S$ ) 与执行价 ( $K$ ) 的差额，即  $K-S$ ，如图 A-15 所示。当标的物价格在一段时间内上涨，期权买入者不能行使其选择权。买入看跌期权在预期价格下跌的前提下表现为有限的风险和无限的利润。

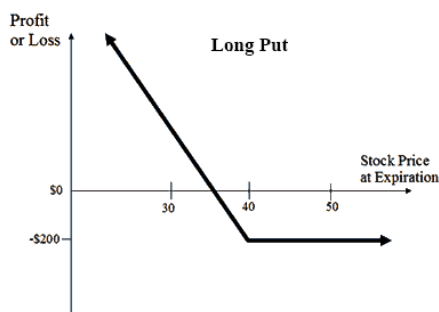


图 A-15 买入看跌期权

### 4. 卖出看跌期权 (Short Put)

获得权利金，一旦标的物价格在未来一段时间内下跌，期权买入者可以履行看跌期权，此时其损失为到期日价格 ( $S$ ) 与执行价 ( $K$ ) 的差额，即  $-(K-S)$ ，或者  $S-K$ ，如图 A-16 所示。当标的物价格在一段时间内上涨，期权买入者不能行使其选择权，赚的就是权利金。卖出看涨期权在预期价格上涨的前提下表现为有限的利润和无限的风险。

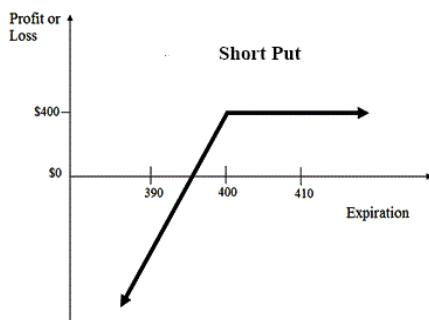


图 A-16 卖出看跌期权

## A.3.2 期权的分类

### 1. 美式期权与欧式期权

一般而言，期权根据实行方式的不同可以分成欧式期权、美式期权、百慕大式期权。欧式期权是仅仅到日期才可以行使的，美式期权是在到期日内任何一天都可以行使的，而百慕大式期权可以被视为美式期权与欧式期权的混合体，比如，期权可以有 3 年的到期时间，但只有在 3 年中每一年的最后一个月才能被执行，它的应用常常与固定收益市场有关。百慕大式期权并非交易所规定的标准化期权，它作为特异期权只能在场外交易。

中国是 2015 年 2 月才在交易所开通期权的，到 2019 年为止发展了 7 个品种的期权，分别如下所示。

- 上证 50ETF 期权：国内唯一的股指期权，是由上交所推出的。例如 510050C1903M02650，合约代码的意思是标的物为 50ETF（510050）的看涨期权（C）到期时间为 2019 年 3 月（1903），合约未发生过除权除息的调整（M），行权价格为 2.65 元（02650）。每张期权合约对应 10000 份“50ETF”基金份额。在这里需要补充，M 是代表合约未发生过除权除息的调整。合约除权除息“调整加挂”后，“M”就会变动，第一次调整为“A”，第二次调整为“B”，依此类推。
- 豆粕期权：商品期权，由大商所推出。例如 m1905-C-2400，合约代码的意思



是标的物为 2019 年 5 月到期的豆粕期货 (m1905) 的看涨期权 (C), 行权价格为 2400 元/吨 (2400), 合约单位为 10 吨的豆粕期货合约。

- 玉米期权: 商品期权, 由大商所推出。例如 c1905-C-1800, 合约代码的意思是标的物为 2019 年 5 月到期的玉米期货 (c1905) 的看涨期权 (C), 行权价格为 1800 元/吨 (1800), 合约单位为 10 吨的玉米期货合约。
- 白糖期权: 商品期权, 由郑商所推出。例如 SR905C4700, 合约代码的意思是标的物为 2019 年 5 月到期的白糖期货 (SR905) 的看涨期权 (C), 行权价格为 4700 元/吨 (4700)。合约单位为 10 吨的白糖期货合约。
- 棉花期权: 商品期权, 由郑商所推出。例如 CF905C15600, 合约代码的意思是标的物为 2019 年 5 月到期的棉花期货 (CF905) 的看涨期权 (C), 行权价格为 15600 元/吨 (15600)。合约单位为 5 吨的棉花期货合约。
- 铜期权: 商品期权, 由上期所推出。例如 cu1905C48000, 合约代码的意思是标的物为 2019 年 5 月到期的铜期货 (cu1905) 的看涨期权 (C), 行权价格为 48000 元/吨 (48000), 合约单位为 5 吨的铜期货合约。
- 天然橡胶期权: 商品期权, 由上期所推出。例如 ru1905C12000, 合约代码的意思是标的物为 2019 年 5 月到期的天然橡胶期货 (ru1905) 的看涨期权 (C), 行权价格为 12000 元/吨 (12000), 合约单位为 10 吨的天然橡胶期货合约。

其中, 股指期货是欧式期权, 商品期权为美式期权。之所以商品期权为美式期权, 是因为考虑到大多数商品期货合约在到期前一两个月 (也就是期权到期时) 成交不活跃, 这样是有助于矫正期权与期货价格的偏离。一旦权利金偏离于期货价格的正常关系, 随时行权有助于矫正这一偏离; 二是降低集中到期行权对标的市场运行的影响。

## 2. 实值期权、平值期权和虚值期权

按照标的价格和执行价格的关系, 期权可分为实值期权、平值期权和虚值期权。期权是非线性衍生工具, 其价值由内在价值和时间价值这两部分组成。内在价值取决于标的物 (国内为上证 50 期货, 白糖期货和豆粕期货) 的价格相对行权价的涨跌, 而时间价值与到期日密切相关, 它是随着持有时间增长而衰减的。实



值期权、平值期权和虚值期权与其内在价值和时间价值有着非常密切的关系。

实值期权（In-the-money, ITM）是指具有内在价值的期权。当看涨期权的执行价格低于当时标的物市场价格时，即  $S > K$ ，该看涨期权具有内在价值，通过行使看涨期权可以获利。同理，看跌期权的执行价格高于相关期货合约的当时市场价格时，即  $S < K$ ，该看跌期权具有内在价值。实值期权的时间价值等于期权总价值减去内在价值。

平值期权（At-the-money, ATM）是指期权的行权价格等于当时的标的物市场价格，即  $S = K$ ，买方立即执行不赢不亏，所以其内在价值为零，平值期权的所有价值均为时间价值，随着到期日临近，时间价值会逐日衰减。期权处于平值时，期权向实值还是虚值转化，方向难以确定，转为实值则买方赢利，转为虚值则卖方赢利，故投机性最强。

虚值期权（Out-the-money, OTM）又称价外期权，同样不具有内在价值。若是看涨期权，则虚值期权表示为执行价格高于当时标的物市场价格，即  $K > S$ ，所以看涨期权不会被行使，期权买入损失权利金。若是看跌期权，则虚值期权表示为执行价格低于当时标的物市场价格，即  $K < S$ 。因为其内在价值为零，虚值期权的所有价值都是时间价值。又由于虚值期权更难向实值期权转化，即难以行权，所以其时间价值也低于平值期权的时间价值。

### A.3.3 期权的影响因素

期权的价值由 5 个因素影响，分别是标的价格、执行价格、到期时间、隐含波动率和无风险利率，每一个因素的改变都会导致期权价格的改变。下面以看涨期权为例，详细介绍这 5 个因素。

#### 1. 标的价格（S）

标的价格上涨会增加看涨期权价值，但它们的关系并非线性，即标的上涨 1%，看涨期权 Call 可能会上涨 1.5%，期权上涨由两部分组成，Delta 和 Gamma。

- Delta 衡量标的资产价格变动时期权价格的变化幅度，当标的价格上涨  $s\%$  时，Delta 对看涨期权 Call 的贡献是增加  $s \times \text{Delta}$ 。
- Gamma 衡量标的资产价格变动时 Delta 的变化幅度，期权价格变动相对于标的物价格变动的二阶导数。Delta 和 Gamma 的关系类似于高中物理课堂上学到的速度和加速度的关系。当标的价格上涨  $s\%$  时，Gamma 对看涨期权 Call 的贡献是增加  $0.5 \times s^2 \times \text{Gamma}$ 。

## 2. 执行价格 ( $K$ )

执行价格上涨，会降低看涨期权 Call 的价格，即随着  $K$  的上升，Call 的类型会逐渐从实值期权 (ITM) 过渡到平直期权 (ATM)，再到虚值期权 (OTM)。需要注意的是，执行价格是合约一开始就定下来的，不存在执行价格变化的风险。

## 3. 隐含波动率 ( $\sigma$ )

标的价格波动增大会增加看涨期权的价值，因为波动率越大，代表标的物可能上涨得越快，因此能获利越多。Vega 衡量标的资产价格波动率变动时期权价格的变化幅度，当标的价格波动率上涨  $\sigma$  时，Vega 对看涨期权 Call 的贡献是增加  $\sigma \times \text{Vega}$ 。一般而言，远月合约流动性不足，敏感度更大，Vega 适用于远月合约操作。

## 4. 无风险利率 ( $r$ )

利率改变也会影响期权价格。Rho 衡量利率转变对期权价格变化幅度。由于中国是利率管制国家，所以利率变化风险可以忽略不计。

## 5. 到期时间 ( $T - \tau$ )

期权存在时间价值，随着到期日临近，该价值会逐渐衰减。Theta 衡量时间变化对期权理论价值的影响，表示时间每经过一天，期权价值会损失多少。

图 A-17 可以比较直观地看到希腊值在期权中的符号表示和含义。



希腊字母	符号表示	含义
Delta	$\Delta$	标的资产价格变化引起期权价格变化
Gamma	$\Gamma$	标的资产价格变化引起 Delta 值的变化
Theta	$\Theta$	期权的时间价值随时间流逝损耗的速度
Vega	$\Lambda$	隐含波动率变化引起的期权价格变化
Rho	$\rho$	期权价格对(无风险)利率变化的敏感程度

图 A-17 5 个希腊值在期权中的符号和含义

期权的 4 种交易形式对应着不同的希腊值，如图 A-18 所示。

	Delta	Gamma	Theta	Vega
买入看涨期权 (Long Call)	+	+	-	+
卖出看涨期权 (Short Call)	-	+	-	+
买入看跌期权 (Long Put)	-	-	+	-
卖出看跌期权 (Short Put)	+	-	+	-

图 A-18 期权的 4 种交易形式对应着不同的希腊值

## A.3.4 期权投资组合

### 1. 跨式组合期权

跨式组合期权 (Straddle) 由具有相同执行价格、相同期限的一份看涨期权和一份看跌期权组成。跨式组合分为两种：底部跨式组合和顶部跨式组合。前者由两份多头组成，后者由两份空头组成，如图 A-19 所示。

$$\begin{aligned}\text{Long Straddle} &= \text{Long Call (K1)} + \text{Long Put (K1)} \\ \text{Short Straddle} &= \text{Short Call (K1)} + \text{Short Put (K1)}\end{aligned}$$

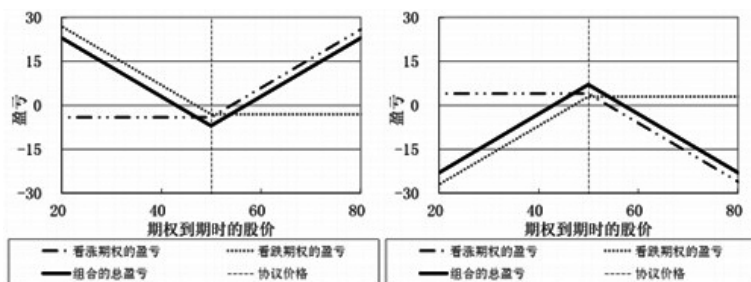


图 A-19 跨式组合期权

## 2. 宽跨式组合期权

宽跨式组合期权（Strangle）由到期日相同但执行价格不同的一份看涨期权和一份看跌期权组成，其中看涨期权的执行价格高于看跌期权。宽跨式组合也分底部和顶部，底部由多头组成，顶部由空头组成。

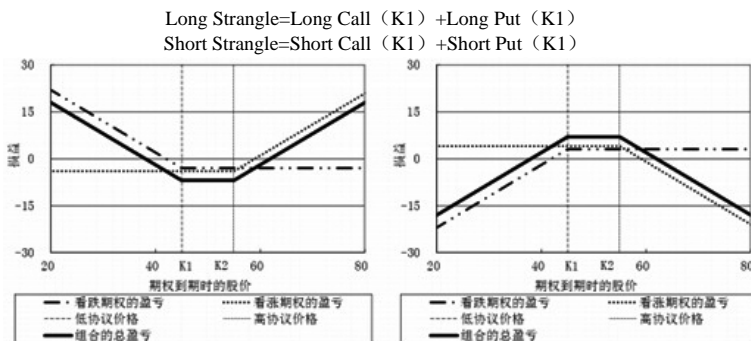
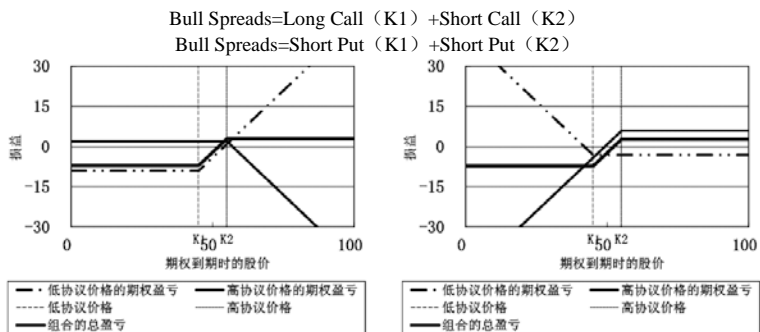


图 A-20 宽跨式组合期权

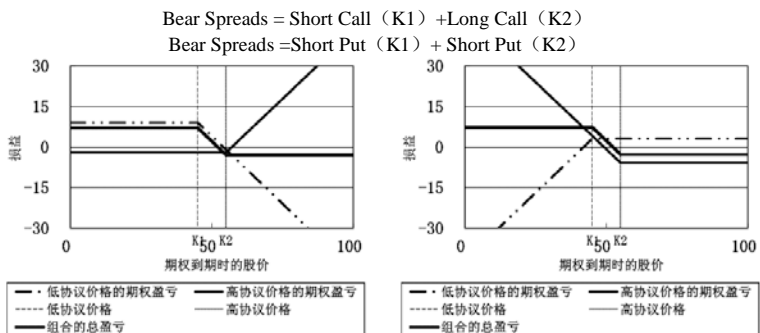
## 3. 牛市差价组合期权

牛市差价组合期权（Bull Spreads）是指由相同到期期限，一份看涨期权多头与一份同一期限较高执行价格的看涨期权空头，或者是一份看跌期权多头与一份同一期限较高执行价格的看跌期权空头构成。到期日现货价格升高对组合持有者较有利，故称牛市差价组合期权，如图 A-21 所示。牛市差价组合策略限制了投资者当股价上升时的潜在收益，也限制了股价下跌时的损失。



#### 4. 熊市差价组合期权

熊市差价组合期权 (Bear Spreads) 是指由相同到期期限, 可以由一份看涨期权多头和一份相同期限、执行价格较低的看涨期权空头组成, 也可以由一份看跌期权多头和一份相同期限、执行价格较低的看跌期权空头组成, 如图 A-22 所示。显然, 到期日现货价格降低对组合持有者较有利, 故称熊市差价组合。



#### 5. 蝶式差价组合期权

蝶式差价组合期权 (Butterfly Spreads) 是由 4 份具有相同期限、不同执行价格的同种期权头寸组成的。其中的一种典型组合为这 4 份期权头寸里共有 3 个执行价格,  $K1 < K2 < K3$ , 且  $K2 = (K1 + K3) / 2$ , 则相应的蝶式差价组合有如下 4 种。

- 看涨期权的多头蝶式差价组合，它由执行价格分别为  $K_1$  和  $K_3$  的看涨期权多头和两份执行价格为  $K_2$  的看涨期权空头组成，其损益分布如图 A-23 左上所示。
- 看涨期权的空头蝶式差价组合，它由执行价格分别为  $K_1$  和  $K_3$  的看涨期权空头和两份执行价格为  $K_2$  的看涨期权多头组成，其损益分布如图 A-23 右上所示，与左上图相反。
- 看跌期权的多头蝶式差价组合，它由执行价格分别为  $K_1$  和  $K_3$  的看跌期权多头和两份执行价格为  $K_2$  的看跌期权空头组成，其损益分布如图 A-23 左下所示。
- 看跌期权的空头蝶式差价组合，它由执行价格分别为  $K_1$  和  $K_3$  的看跌期权空头和两份执行价格为  $K_2$  的看跌期权多头组成，其损益分布如图 A-23 右下所示。

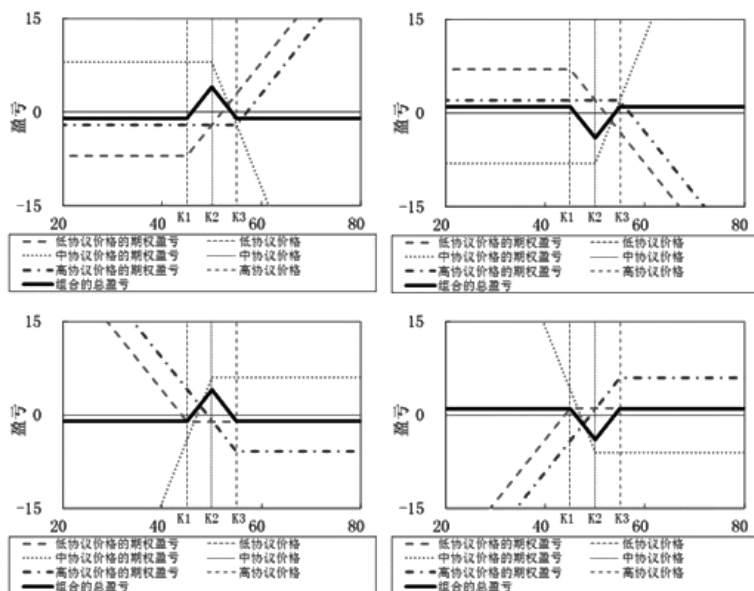


图 A-23 蝶式差价组合期权

### A.3.5 期权波动率套利策略

期权交易策略主要分为两大类：交易标的方向和交易波动率，也就是常说的赌方向和赌波动率。

赌方向一般不用考虑希腊值（Delta，Gamma，Theta 和 Vega），但是会暴露更大的风险。例如买入看涨期权或看跌期权来做一个方向性的交易，Gamma 基本上没有什么用，因为这时候你肯定已经很清楚你买入了正值的 Gamma。这个正值的 Gamma 在标的资产价格上涨时会增加看涨期权的 Delta，在标的资产价格下跌时会增加看跌期权的 Delta。正值的 Gamma 会使得期权头寸在越来越实值的过程中，Delta 越来越大，从而增加你的赢利能力。简单地说，在方向性的交易中，正值的 Gamma 在你赚钱的时候会让你加速赚钱，在你亏钱的时候会让你减速亏钱。

交易波动率对于交易方向的优势就是更低的风险和更大的收益。大量的学术研究表明，股票的价格基本属于随机游走的状态，波动率是标的物对数收益率的方差，而且存在均值回归，即可以通过研究历史隐含波动率来预测未来波动率的大小。一句话总结就是标的物是涨是跌太难猜了，但波动率变化更好猜。波动率交易比方向性交易的胜率更大。

波动率套利策略，其收益不依赖于标的资产的价格变动方向，而依赖于标的资产的价格波动情况，其核心是寻找期权的隐含波动率和市场的实际波动率的价差，并对其进行相应交易，换句话说如果预测的波动率与期权的隐含波动率有显著不同，可以通过相应的期权交易进行获利。具体操作是通过期权简单组合来对冲某一个希腊值，通过对波动率的预判来管理好其他希腊值来达到赢利的目的。

若通过对历史数据的统计判断隐含波动率会上升，则采取做多波动率策略，通过买入近月平值看涨期权和近月平值看跌期权构建跨式组合期权，对冲期权组合的 Delta 值，即让投资组合 Delta 中性，Gamma 和 Vega 可以增加期权价值，Theta 会减少期权价值，隐含波动率上升导致投资组合的整体价值上升，获利离场，通过标的物（期货）来对冲那部分额外的价值，让投资组合回归 Delta 中性。



## A.4 外汇

### A.4.1 外汇的定义

外汇是外国货币或者以外国货币表示的能用于国际结算的支付手段。外国货币不一定是外汇，主要看其能否自由兑换，或者这种货币能否重新回流到它自己的国家，可以不受限制地存入该国的任意一家商业银行的普通账号上，而且在需要时可以任意转账。因此，主流外汇品种并不多，在外汇市场上进行交易的主要为美元（USD）、欧元（EUR）、日元（JPY）、英镑（GBP）、瑞士法郎（CHF）、加元（CAD）和澳元（AUD）。

外汇市场是全球最大的金融市场，其市场容量远超股票市场、债券市场和期货市场。据国际清算银行（BIS）2016年9月发布的三年一次的调查报告 *Triennial Central Bank Survey: Foreign exchange turnover in April 2016*，外汇市场日均交易量达 5.1 万亿美元，其中美元是最大的交易货币，占据了交易总额的 87.6%；欧元则是第二大交易货币，其交易份额为 31.3%；日元排名第三，占 21.6%，如图 A-24 所示。

最活跃的货币对分别是欧元/美元（占 23%）、美元/日元（占 17.7%）、英镑/美元（占 9.2%）、澳元/美元（占 5.2%）和美元/加元（占 4.3%）。



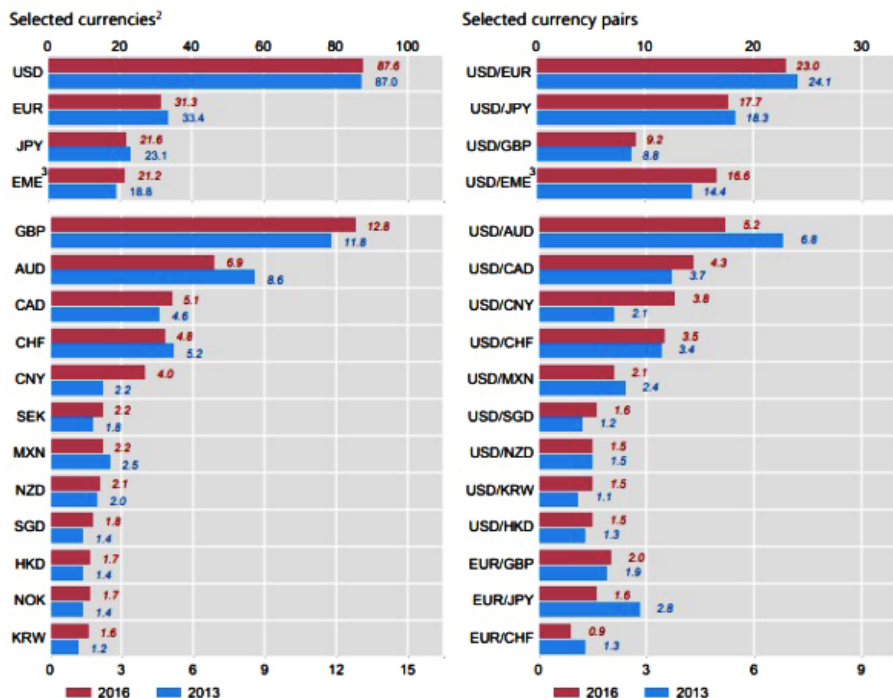


图 A-24 国际清算银行关于外汇市场的报告

## A.4.2 外汇市场的结构

由于外汇市场的规模是全球最大的，其市场参与者相对于其他市场是最为全面的。按照交易主体来分类，市场参与者可分为以下几类。

### 1. 中央银行

中央银行（以下简称中行）是外汇市场最重量级的玩家，也是外汇市场流动性的主要提供商，例如美联储、欧洲央行、人民银行、英格兰央行等。央行参与外汇市场并不以赢利为目的，而是为了维持汇率稳定、调节利率和货币投放量，通过货币政策来协助政府更好地进行宏观调控。为了使汇率维持在一定的水平上，央行通常设立外汇平准基金，当市场上外汇供不应求时，汇率下跌，央行需要抛售外币，购买本币。反之若外汇供过于求，汇率上涨，央行就会买进外币，投放

本币。

## 2. 商业银行

商业银行是第二大的流动性提供商，其自身存款、贷款、汇兑、储蓄等业务会吸收庞大机构客户的外汇兑换订单，需要通过在外汇市场的交易来满足客户的换汇需求。在外汇领域比较著名的商业银行有花旗银行、汇丰银行、德意志银行和星展银行。

## 3. 投资银行

投资银行指的是主要从事证券发行、承销、企业重组、兼并与收购、投资分析等业务的非银行金融机构，例如高盛公司、摩根士丹利、瑞士银行和中信证券等。其 FICC（固收、外汇及大宗商品）业务占到投行业务收入的 50% 以上。外汇业务的客户通常是高频交易公司、对冲基金和跨国企业等。投行有义务为机构客户提供双边报价，如某对冲基金要做多 100 亿日元，投行会在它的资产负债表中把 100 亿日元卖给基金公司，同时为了消除暴露出来的头寸，投行会可以在银行间进行交易，买入同等额度的日元，其赢利来自买卖价中间的点差。

## 4. 跨国企业

跨国企业由于在世界各地设立众多分支机构或子公司，从事生产经营活动，在赚取各国货币后，需要定期换汇成母国货币汇入总公司，于是也成为全球外汇市场流动性消费商。美国可口可乐公司在全球 200 多个国家经营 160 种饮料品牌，其赢利产生巨额的欧元、人民币、墨西哥比索、印度卢比，乃至更加小众的巴西雷亚尔、阿根廷比索等，都需要通过银行换成美元汇入母公司。

## 5. 高频交易公司

相对于商业银行和投资银行，高频交易公司属于非义务做市商，并非每时每刻都提供买卖双边报价，可能在极端行情下它会关闭买家或者卖家。顾名思义，高频交易公司的报价刷新频率远高于银行机构，对外汇市场最大的贡献就是更快速地撮合交易，使得成交的时间大幅度缩短。不像银行机构，高频交易公司的资



金流不多，其赢利的核心是速度和算法。比较有名的高频交易机构有 KCG、Tower Research Capital、Jump Trading 和 Virtu Financial 等。

## 6. 投资基金

投资基金包括投资全球股票、债券和期货品种的公募基金，以及国家政府利用外汇储备资产与国家财政盈余在全球范围内进行投资的主权财富基金。与跨国企业一样，他们属于流动性消费商，他们换汇的目的是对全球股票、债券、期货等市场进行全球资产配置，以获得相对稳定的收益，而不追求外汇本身涨跌带来的收益。大型的投资基金有挪威的政府全球养老基金、中国的中投公司、贝莱德基金和富达基金等。

## 7. 对冲基金

上面介绍的都是大型的做市商机构，它们主要是为市场提供流动性，并不进行投资，而对冲基金则是外汇市场上投资的中坚力量。对冲基金对外汇并没有真实的需求，如调整头寸或清偿债权/债务，他们参与外汇买卖纯粹是为了寻找因市场障碍而可能利用的获利机会，预测汇价的涨跌，以做多或做空的形式，根据汇价的变动低买高卖，赚取差价。有时候，在足够大的利益驱动下，对冲基金可以操纵大量的巨额资金，对某种币种顺势发动突然袭击，影响某种货币的正常趋势，加剧外汇市场的动荡。“金融巨鳄”索罗斯的量子基金，在 1992 年袭击英镑，净赚 10 亿美元，在 1997 年亚洲金融风暴中，又袭击泰国、马来西亚等国的货币，又有百亿美元入账，后又闪袭港币，但遭遇惨败，于 2012 年做空日元，揽近 12 亿美元。

## 8. 外汇经纪商

外汇经纪商在外汇市场仅扮演着中介人的角色，以赚取点差或者佣金为目的，为客户代洽外汇买卖的汇兑商定，在买主与卖主间拉拢撮合，透过外汇经纪人的接洽，直接或间接自银行买卖。外汇经纪商可以分为以下两种。

一种是机构外汇经纪商，这里的机构指的是市场流动性提供方，或者说做市

商机构，例如央行、商业银行、投资银行、高频交易公司等。常用的经纪商平台有两个：一个是毅联汇业（ICAP）旗下的 EBS 电子交易平台，EBS 是银行间外汇市场使用最广的平台，在世界 50 个国家的 800 多家交易所为银行和非银行交易机构提供即期外汇交易定价系统；另一个是路透社旗下的 Reuters 3000 Xtra。定位是顶尖银行的 ECN 系统，占据了外汇市场现货交易近 70% 市场份额，小型机构投资者和散户是接触不到的。

第二种经纪商指的是零售外汇经纪商，它是散户进入外汇市场的唯一途径。过去外汇市场是机构投资者的天下，每交易 1 手可能需要 100 万美元以上，而零售外汇经纪商的作用可以让传统意义上非常小的玩家参与到全球外汇市场上。现在存在 4 种零售外汇经纪商模式，分别是零售电子化通信网络平台（零售 ECN）、做市商模式（MM）、直通式助理模式（STP）和多边交易设施模式（MTF）。零售 ECN 平台与后三者不同，它收取的是佣金而不是点差，同时散户在 ECN 上可以得到更好的价格，与之对应的，它的开户门槛也是最高的，一般需要 10 万美元的资金量。

## 9. 个人投资者

个人投资者或者说散户，处于食物链的底端，他们通过零售外汇经纪商进入全球外汇市场，面临的风险是差价合约（CFD）的高杠杆、低风险承受能力、高点差导致的赢利困难，以及最应该注意的出金风险。在我国内地并无合规的零售外汇经纪商，散户接触的平台都是在中国香港开设的公司，在信息不透明的情况下被网上虚假广告轰炸，会让新手非常容易接触黑平台或者在黑平台开户。黑平台有非常高的破产风险或者卷走资金跑路的风险。

散户进入市场的首要目标就是保证其资金的安全，第一步就是选择可靠的零售外汇经纪商。一般正规的、大型的平台并不会在网上打广告，也不会开户给赠金，或者非农日入金送赠金，对于国内个人投资者，比较有名的零售外汇经纪商有盈透证券（MTF）、福汇（STP+ECN）、Oanda（MM）等。



### A.4.3 外汇市场的组织形式

外汇市场的规模是金融市场中最大的，其市场风险和信用风险与我们熟知的股票市场和期货市场是不一样的。外汇价格每一跳（Tick）的市场风险可能包含几百万美元的信用风险，而股票期货合约规模就小很多，合约价格每一跳对应的可能是几十美元的信用风险。这会导致集中清算的成本非常高，因此外汇市场上很难出现集中清算的交易所，外汇交易的主要形式是场外交易（OTC）。

外汇具有全球性，而且市场参与者角色复杂，他们各自的信用风险和破产风险都不同，央行的信用等级最高，零售经纪商和散户的违约风险最大，因此大型银行通过 EBS 或者 Reuters 3000 Xtra 进行银行间交易，而小型机构和个人投资者只能通过零售经纪商进行交易。

国际清算银行（BIS）2016 年调查报告指出，在日间交易量为 5.1 万亿美元的外汇市场中，银行间交易占总交易量的 42%，即仅仅靠银行同业拆借就达到了 2.1 万亿美元的交易量，所以各国的央行、商业银行是外汇市场的大玩家，如图 A-25 所示。而大型银行与非金融公司，一般指的是跨国企业，如可口可乐公司和麦当劳等，它们之间的交易占 7%，即 3570 亿美元的成交量。

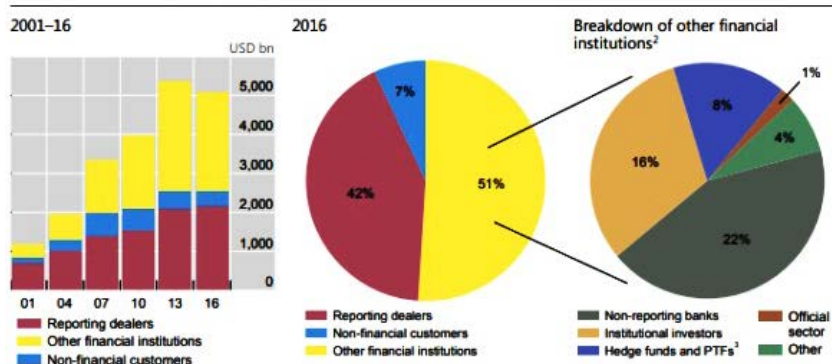


图 A-25 外汇市场的组织形式

剩下的 53% 是超级做市商与其他非银行机构之间的交易，这里的“其他”指的是投资银行、高频交易公司、对冲基金等。其中 22% 是超级做市商投资银行或

者中小型商业银行的交易，16%属于与基金公司发生的换汇业务，8%指的是对冲基金和高频交易公司，而最后的4%就是市场上更小的玩家，如小型进出口商和个人投资者等。

## A.4.4 主要外汇交易中心

世界上大概有30个主要外汇交易中心，它们分布于世界各大洲的不同国家和地区，其中，最重要的有欧洲的伦敦、法兰克福、苏黎世和巴黎、美洲的纽约和洛杉矶、大洋洲的悉尼、亚洲的东京、新加坡和中国香港。

按照交易量，英国伦敦无疑排在首位，其日均交易额高达2.4万亿美元，这里群英荟萃，很多在外汇领域大型的银行或者对冲基金都把总部或者外汇交易部门设在伦敦。然后是美国纽约，它贡献的交易量也达到每天1.3万亿美元，第三到第五名分别是新加坡、中国香港和日本东京，这3个亚洲外汇交易中心合计1.4万亿美元。总体来说，欧洲的市场容量最大，然后是美洲，最后才是亚洲市场与大洋洲市场。同时由于欧洲与美洲市场相差不大，在中间某个交易时段欧美市场产生重叠，市场活跃度最大。图A-26展示的是世界主要外汇市场交易时间表。

5:00—14:00 行情表现较为清淡，主要由于亚洲市场的推动力量较小，一般震荡幅度在30点以内，没有明显的方向，多为调整和回调行情。由于市场活跃度不高，捕捉行情波动的产生赢利不易覆盖点差的成本，投机主体参与度不高。在该时段交易需要关注日本经济要员对于日元走势的言论、日本贸易收支、宏观经济报告、全国百货店零售额等指标，这会对日元波动起到较大影响。





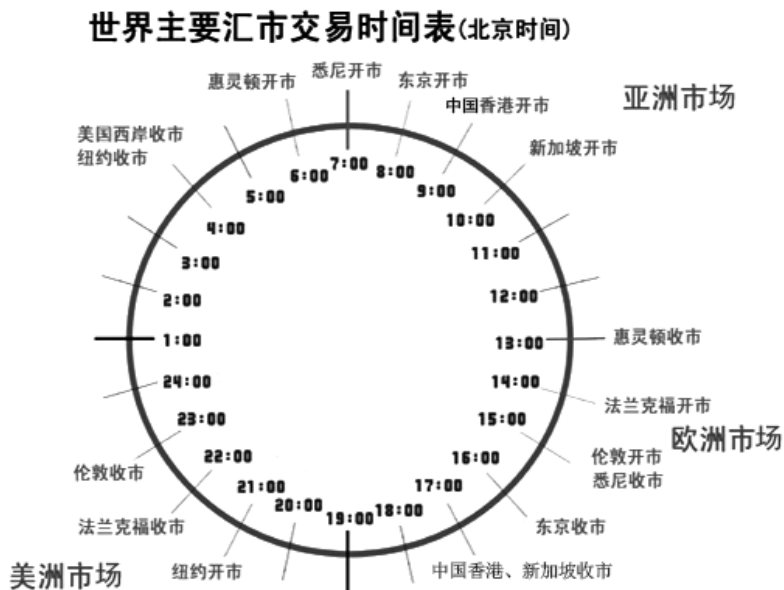


图 A-26 世界主要外汇市场交易时间表

14:00—20:00，欧洲上午市场，特别是在 15 点后伦敦开市后，欧洲市场开始活跃起来，该时段也会伴随着一些对欧洲货币有影响力的数据的公布，如欧洲央行利率决议、德国 IFO 景气判断指数、欧元区 12 国 GDP、欧元区 12 国消费者价格指数等，也将对欧洲外汇市场波动程度有影响，产生一定程度的市场波动，震荡幅度在 40~80 点左右。一些投机力量也会像闻到血腥味的小鱼一样相继进场。傍晚 18:00—20:00 欧洲市场的中午休息时间，同时也是美洲市场的清晨，等待美洲市场开市，市场行情走弱。

20:00—24:00，欧洲市场的下午盘和美洲市场的上午盘，这段时间是行情波动最大的时候，也是资金量和参与人数最多的时段，震荡幅度可以达 80 点以上。该时段不仅仅关注欧洲新发布的经济数据指标，也需要注意美国的最新基本面数据公布，如美国非农就业数据、GDP 数据、公开市场委员会利率、美联储公开言论、生产价格指数、消费价格指数等一系列基本面数据，都会成为影响汇市的重要因素。

00:00—5:00，美洲市场的下午盘，一般此时已经走出了较大的行情，此段时间多为对前面行情的技术调整。



## A.4.5 外汇交易策略

### 1. 全球宏观策略

全球宏观策略就是通过对全球经济宏观面的预测，从上至下的方式来筛选并进行外汇、期货、股票等标的的交易。全球宏观策略的投资方法，主要通过对股票、货币、利率以及商品市场的价格波动进行杠杆押注，来尝试获得尽可能高的正投资收益。这个定义中的“宏观”一词，来自基金经理试图利用宏观经济的基本原理来识别金融资产价格的失衡错配现象，而“全球”则是指可以在全世界的范围内寻找发现这种价格错配的现象。

该策略对某一个国家或者某一个领域的未来一段时间进行预测，可以依靠基金经理的主观判断，也可以用量化的模型把基本面的数据、价格面数据、历史数据、分析师预测加在一起产生一种信号比较强的中长期的预测，持仓周期可以达到 3~4 年。在这个策略的交易过程中，可能交易的重点都不在于信号本身了，而是在于基于信号方向建立交易持仓量的控制，市场交易中对于自己不利价格的风险控制，这个时候就要结合对冲策略与短线交易，比如进行一些外汇期权交易、价差交易。主观全球宏观的代表是“金融巨鳄”索罗斯的量子基金和保罗·都铎·琼斯的都德期货基金，而量化全球宏观比较有名的是 Alpha Simplex Group 及 FX Concept。

全球宏观对冲基金交易可被分为两大类：直接的定向型交易和相对价值型交易。定向型交易指经理们对一种资产的离散价格的波动情况下注，比如做多美元指数或做空日本债券；而相对价值型交易指通过同时持有一对两类类似资产的多头和空头，以期利用一种已被发现的相对价格错位来赢利，比如在持有新兴欧洲股票多头的同时做空美国股票，或者在做多 29 年期德国债券的同时做空 30 年期德国债券。



## 2. CTA 策略

外汇 CTA 策略原理与期货 CTA 策略一样，主要分为趋势跟踪、价差套利。总体来说，CTA 策略在外汇市场上的应用比期货市场少很多，因为该策略在外汇市场比较难赚钱。

趋势跟踪策略主要通过分析外汇品种的历史价格变化进行建模，同时也需要嵌入基于基本面数据的全球宏观模型或者基于更加宏观的视野的主观判断，产生交易信号。

价差套利则更加注重分析国家的产业结构、贸易往来、国与国之间的关系，来赚取外汇币种之前，或者外汇与大众商品之间的价格偏离。例如加拿大是全球重要的石油出口国，加元又被称为石油货币，油价上涨不仅会带动原油指数上涨，也会让加元相对于其他外币强势起来。

CTA 策略的持仓周期可以是几分钟到几天，远远短于全球宏观策略，所以需要考虑新的交易成本，即点差。交易越频繁，对点差的敏感度越高，如何争取更低的点差是策略赢利的关键点之一。

## 3. 高频交易策略

在外汇领域主要是高频做市商。高频交易策略基于时间序列模型和市场微观结构的历史数据进行统计，解析市场价格在未来数秒内的走势，加一个价差并且提供报价，赚取点差。如果市场变动慢，那么做市商的敞口风险也比较低；如果市场变动比较快，高频做市商可能会选择在银行间市场把自己的敞口对冲出去，赚取银行间市场与下游市场之间点差的差值利润，所以高频交易需要追求更低延时的交易系统，最少到微秒级甚至纳秒级的交易速度。

外汇市场属于场外交易市场，缺乏一个集中清算中心，功能相当于小型交易所的 ECN 平台，产生了很多跨市场套利的赢利机会，抢单的速度要够快才能捕捉到稍纵即逝的套利机会。

## 参考文献

- [1] Derman E. 宽客人生：从物理学家到数量金融大师的传奇[M]. 北京：机械工业出版社，2015-01.
- [2] 董可人. 我是高频交易工程师[M/CD].
- [3] 黄婷. 宽客在中国[N]第一财经日报，2012-02-18.
- [4] Poundstone W. 财富公式：玩转拉斯维加斯和华尔街的故事[M]. 沈阳：万卷出版公司，2008-01.
- [5] 忻海. 解读量化投资：西蒙斯用公式打败市场的故事[M]. 北京：机械工业出版社，2010-01.
- [6] 左涛，庞彦广，傅东海，等. 一种高频交易高精度监察系统的技术研究与实践[EB/OL].
- [7] 高道德，袁林青. 2017 年金融工程中期策略——从 “Alpha +Beta” 的角度分析量化产品业绩表现[EB/OL].
- [8] 刘文波，尚芳. 股指期货专题研究之跨品种对冲策略[EB/OL].
- [9] 章顺，李晓辉，田钟泽，等. 商品期货套利实证[EB/OL].
- [10] 安尉，王君，李而实. 中信建投策略[EB/OL].
- [11] 邓璎函. 期权介绍：期权的损益和交易策略[EB/OL].



- [12] 魏强斌. 外汇交易-进价: 从新手到大师的成功之路[M]. 北京: 经济管理出版社, 2014-05.
- [13] 凌乐. 挖掘 T+0 交易策略, A 股有哪些套利缺陷[EB/OL].
- [14] 徐玉宁, 马自妍. CTA 策略巡礼: 从趋势到套利[EB/OL].
- [15] Libertini N. 详解反趋势交易[EB/OL].
- [16] 刘富兵, 陈睿. 期权应用之交易策略篇--期权系列研究报告之十[EB/OL].
- [17] 王勇. 期权策略大爆炸系列专题五: Gamma 中性与 Gamma 中性交易[EB/OL].
- [18] Hull J. 期权, 期货及其他衍生产品[M]. 北京: 机械工业出版社, 2009-01.
- [19] 王焕然. 解析高频交易[EB/OL].
- [20] 蓝海平. 高频交易的技术特征、发展趋势及挑战[EB/OL].